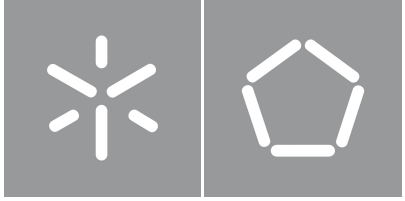
University of Minho
School of Engineering

Daniel Barbosa Miranda

P4-CAVODE, a lightweight anomaly detection system in Software-Defined Networks

October, 2024



University of Minho
School of Engineering

Daniel Barbosa Miranda

P4-CAVODE, a lightweight anomaly detection system in Software-Defined Networks

Master's Dissertation
Masters in Informatics Engineering

Dissertation supervised by
João Marco Cardoso Silva

October, 2024

Copyright and Terms of Use for Third Party Work

License granted to users of this work:

Acknowledgements

I would like to express my gratitude to:

- my supervisor João Marco Silva whose guidance and availability were instrumental during the research and development of this project.
- my colleague Rui Pedro Monteiro, whose shared similar interests and collaborative efforts greatly contributed to the project.
- INESC TEC and FCT for funding and providing me with the appropriate tools.
- my family and friends who provided support and encouragement throughout my academic journey.

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project 10052/BII-E_B4/2023.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, October, 2024

(Daniel Barbosa Miranda)

Abstract

P4-CAVODE, a lightweight anomaly detection system in Software-Defined Networks

As the Internet evolves and its global user base continues to expand, the demand for scalable intrusion detection systems becomes increasingly critical. While scalability is essential, it is also important to recognize that malicious users within this growing user base can employ innovative methods to interact with systems. Consequently, regular updates are necessary to keep up with such methods. In this context, the use of Software-Defined Networks and languages such as P4 proves invaluable, as they allow for modifications to be performed on the data plane logic while avoiding constant hardware upgrades.

This dissertation presents P4-CAVODE, a lightweight network anomaly detection system designed for identifying port scans, a common technique employed during the initial stages of cyberattacks to gather information about a target. P4-CAVODE is able to detect irregularities caused by multiple variations of port scans, as it considers target multiplicity and is independent of the headers used. However, its focus lies in the timeframe used, namely on slow scans which consist of a main challenge due to their extended duration, especially in data planes where available memory is limited.

The developed solution manages to sustain a throughput that closely matches the theoretical capabilities of the hardware, even when using multiple services, which demonstrates its scalability. Furthermore, despite its focus on slow port scanning, deviations caused by port scan probes do not require large periods to be identified. As such, their detection occurs practically in real-time, ensuring that actions against imminent threats could be taken with minimal delay. These characteristics highlight the role of data plane programmability in the realm of cybersecurity.

Keywords: Software Defined Network, Programmable Data Planes, P4, Port Scan, Anomaly Detection

Resumo

P4-CAVODE, um sistema lightweight de deteção de anomalias em Redes Definidas por Software

À medida que a Internet evolui e a sua base global de utilizadores continua a expandir-se, a procura de sistemas de deteção de intrusão escaláveis torna-se cada vez mais crítica. Embora a escalabilidade seja essencial, é também importante reconhecer que os utilizadores maliciosos desta base crescente de utilizadores podem empregar métodos inovadores para interagir com os sistemas. Consequentemente, são necessárias actualizações regulares para acompanhar esses métodos. Neste contexto, a utilização de redes definidas por software e de linguagens como P4 revela-se inestimável, uma vez que permite efetuar modificações na lógica do plano de dados, evitando actualizações constantes do hardware.

Esta dissertação apresenta P4-CAVODE, um sistema *lightweight* de deteção de anomalias de rede projetado para identificar port scans, uma técnica comum empregue durante as fases iniciais de ciberataques para recolher informações acerca de um alvo. P4-CAVODE é capaz de detetar irregularidades causadas por múltiplas variações de port scans, uma vez que considera multiplicidade de alvos e é independente dos *headers* utilizados. No entanto, o seu foco está no intervalo de tempo usado, nomeadamente em *slow scans* que constituem um grande desafio devido à sua longa duração, especialmente em planos de dados em que a memória disponível é limitada.

O algoritmo desenvolvido consegue manter um *throughput* próximo das capacidades teóricas do hardware, mesmo quando utiliza múltiplos serviços, o que demonstra a sua escalabilidade. Além disso, apesar do seu foco em *slow port scanning*, os desvios causados pelas *probes* de *port scan* não requerem grandes períodos para serem identificados. Como tal, a sua deteção ocorre praticamente em tempo real, garantindo que acções contra ameaças iminentes podem ser tomadas com um atraso mínimo. Estas características realçam o papel da programabilidade do plano de dados no domínio da cibersegurança.

Keywords: Redes Definidas por Software, Planos de dados programáveis, P4, Port scan, Deteção de anomalias

Contents

List of Figures

List of Tables

Acronyms

ACWM Adaptative Cumulative Windows Model. 28, 29

ADWIN Adaptative Windowing. 28

API Application Programming Interface. 7

ASIC Application-Specific Integrated Circuits. 7–9

BMv2 Behavioral Model v2. ix, 8, 9, 12, 38, 41, 46, 47, 64, 68

BSD Berkeley Software Distribution. 17

CPU Central Processing Unit. 7, 8

CUSUM Cumulative Sum. 28

DL Deep Learning. 23

DoS Denial of Service. 31

DSL Domain-Specific Language. 3

EWMA Exponentially Weighted Moving Average. 28, 29, 31, 33, 35, 36

EWMMD Exponentially Weighted Moving Mean Deviation. 31

FCWM Fixed Cumulative Windows Model. 28, 29

FPC Flow Processing Core. 8, 9, 65, 68

FPGA Field Programmable Gate Arrays. 7, 8

FTP File Transfer Protocol. 17, 18

Gbps Gigabits per second. 7, 8, 47, 48, 64, 65

HDL Hardware Description Languages. 8

HIDS Host Intrusion Detection System. 21

IANA Internet Assigned Numbers Authority. 14, 15

ICMP Internet Control Message Protocol. 16, 17

ICS Industrial Control System. 30

IDS Intrusion Detection System. 19, 20, 30

IP Internet Protocol. 1, 8, 13, 15, 17–19, 21, 22, 24, 30, 31, 33–37, 43, 45, 46, 50, 58, 68

IPS Intrusion Prevention System. 20, 23, 30

IR Intermediate Representation. 7, 41

LPM Longest Prefix Match. 10

MAU Match Action Units. 7

NFCC Netronome Flow C Compiler. 41

NFP Network Flow Processor. 8, 9, 38, 45

NFV Network Function Virtualization. 4, 38

NIC Network Interface Card. 8

NIDS Network Intrusion Detection System. 3, 20, 29–31, 46

NPU Network Processing Unit. 7, 8

P4 Programming Protocol-independent Packet Processors. iv–vi, 3, 4, 6–11, 26, 29–31, 37, 38, 41, 42, 46, 67

P4-CAVODE P4 Cyclic Adaptative Variation in the Occurrence of Distinct Entries. iv, v, viii, 4, 33, 34, 37, 38, 41, 43–48, 50, 56, 60, 64–68

PCF Partial Completion Filter. 22

PDP Programmable Data Plane. 2, 3, 67, 68

PHT Page-Hinkley Test. 29

RFC Request For Comments. 17

SDN Software-Defined Network. iv, viii, 2, 3, 67

SmartNIC Smart Network Interface Card. ix, 8, 9, 12, 38, 41, 45–47, 64, 65

SRAM Static Random-Access Memory. 11

Tbps Terabits per second. 7

TCAM Ternary Content-Addressable Memory. 11

TCP Transmission Control Protocol. 1, 13, 15–18, 21, 22, 24, 47

UDP User Datagram Protocol. 15, 17, 18

Chapter 1
Introduction

With the continuous evolution of technology, driven by people's increasing demands in fields such as entertainment, communication, education, work, utility services and more, there is an undeniable push for networks to expand and evolve. However, with this expansion comes an escalating threat of cyberattacks, which aim to compromise the integrity, confidentiality and availability of information within a network. These attacks may come in many forms, however, what they usually have in common in their workflow is an initial phase where information concerning a target is acquired. This is known as *reconaissance* [69, 93]. A technique commonly employed during this stage is the port scan [6], which consists of checking a range of transport layer ports in the TCP/IP model on one or multiple hosts to determine their state. This is done by sending specially crafted packets, known as probes, to obtain responses from the ports, which may indicate their state.

While this act is not inherently malicious and can be used to safeguard a system against some of its own vulnerabilities, attackers may seek to use it to discover open ports and exploit vulnerabilities in the underlying services or applications [9]. In an attempt to minimize the likelihood of being detected by a detection system, this technique may suffer multiple variations, regarding factors such as headers, multiplicity of its source or target, or even the period over which it takes action.

A particular challenge among these variations is the slow scan. While in a common port scan, typically known as a fast scan, the ports are probed in a low time period, the slow scan has an increased interval between probes, so that stateful detection solutions may not be able to maintain information long enough to make the correlation between those probes. While a port scan is not malicious per se, it may be an early sign of an imminent threat. As such, identifying its occurrence might help to protect against it beforehand. Variations such as the ones described are not exclusive to port scanning, highlighting the need for cybersecurity solutions to be constantly updated in order to match the ongoing creativity of hostile entities.

This is where traditional network limitations in factors like flexibility and efficiency while maintaining performance become increasingly apparent and challenging to overcome. In traditional network archi-

tectures, both the control plane, responsible for packet routing, and the data plane, tasked with packet forwarding are integrated within a single network device. Moreover, the processing of packets within the data plane is mainly determined by the configurations preset in the hardware equipment, implying that the protocols employed and the forwarding behaviors are installed by the device manufacturers [61]. This integration significantly hinders the introduction of new functionalities and protocols, as new devices and consequently increased costs would be required whenever such updates or modifications are to be performed [60].

One effective approach to address this issue involves decoupling the control plane from the data plane and centralizing the control logic of multiple devices into a central Controller. This notion is the foundation of Software-Defined Network (SDN) and Figure 1.1 illustrates the differences between the discussed network architectures.

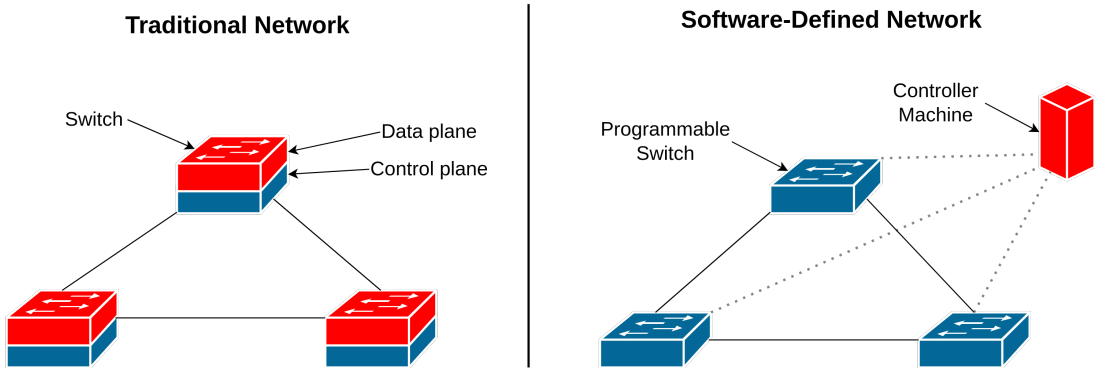


Figure 1.1: Traditional network and SDN architectures [56]

Here, `OpenFlow` [24] arose as the most prominent protocol by enabling communication between the devices where each of the aforementioned planes resides. While it allows for better use of resources when compared to traditional networks, it still presents certain limitations, such as possessing a fixed set of headers and actions that the devices can perform, thus not being able to fully customize the behavior of data forwarding on network devices (e.g., switches and routers) [33].

These constraints regarding customization pointed to the need for a more adaptable and dynamic approach to network management, known as Programmable Data Plane (PDP). This concept brings different advantages in terms of flexibility, by allowing the addition or removal of features and protocols, or even correction of errors through simple software updates, thereby leading to potential cost savings [61]. Moreover, PDPs offer greater performance and scalability due to the fact that the data plane can be optimized for specific workloads and environments, leading to a faster process and a lower latency, which can be beneficial for applications that require real-time communication and high throughput [60, 61]. Considering the separation of the planes, a control plane can look over multiple dataplanes easing out

the monitoring process. On top, tasks that were previously the responsibility of the control plane can be transferred to the data plane, improving overall efficiency [52].

Considering these aspirations, the Domain-Specific Language (DSL) Programming Protocol-independent Packet Processors (P4) [90] stands out by being able to fill this role, thanks to its high expressibility through a large set of constructors and operators that facilitate the personalization of packet processing within the data plane. Despite that, the P4 language should not necessarily be seen as a substitute for the OpenFlow protocol, as they can be used together since the former allows the behavior of network devices to be defined while the latter can serve as a communication interface between them [33].

1.1 Problem

Network communication involves both legitimate participants who intend to access services and malicious ones who seek to harm the system or other users, thus it becomes necessary to distinguish their performed activities. This is usually done using Network Intrusion Detection Systems (NIDSs). In this respect, PDPs provide great flexibility and responsiveness by allowing the creation and adaptability of configurations against new threats from which the structure should be protected. The simplicity and expressiveness of P4 enable configurations to be implemented in a more general way, enhancing support and portability at the hardware level [51]. Furthermore, because the data plane is closer to the hardware, acquiring statistics pertinent to intrusion detection becomes more straightforward, thereby possibly accelerating how a system could react to a threat [67].

Despite that, solutions developed with SDN in mind for addressing port scan techniques are scarce, even more so when it comes to detecting slow scans. Thanks to its characteristics, it usually requires the usage of stateful structures to correlate its probes, which not only can be difficult to adjust in terms of thresholds of how long data is kept but also require a larger memory footprint, something that a data plane might have trouble enduring. As such, most state-of-the-art solutions, regarding detection systems in SDN, not only do not consider slow port scan but also are not tested in actual hardware.

1.2 Objectives

The objective of this dissertation is to showcase the potential of utilizing PDPs within network devices, specifically in the field of cybersecurity. It focuses on the development of a network system capable of detecting anomalies caused by slow scans, without dismissing the capabilities of detecting the remaining varieties of scans. Slow scans by themselves are challenging given their time window. Most existing

solutions concerning it use stateful structures that keep information regarding the packets for a relatively long time period, so that they may be able to make a correlation between them and conclude that it is suspicious. However, when pertaining to data plane programmability, the system might not be able to withstand a high memory footprint, therefore, the developed solution has to have lightweight properties, namely in terms of time and space complexity, and scalability.

The proposed solution should be implemented using the P4 language and be capable of operating on both software environments where network devices are emulated, hence making it function as a Network Function Virtualization (NFV), and in hardware devices. To achieve this purpose, it is necessary to:

- Understand the functioning, capabilities and limitations of the P4 language;
- Explore both intended software and hardware targets and their underlying differences that might affect portability;
- Infer variations to which the port scan technique might be subjected to;
- Assess existing intrusion detection methods;
- Research data structures that may result in a reduced footprint;
- Analyze algorithms that detect deviations in data streams;
- Define the requirements to detect anomalies caused by port scans and conceptualize a solution capable of addressing these requirements;
- Create adequate test environments and scenarios to evaluate crucial aspects expected from such a solution while also taking into consideration how it affects the system in terms of throughput.

1.3 Document structure

Chapter 2 begins by introducing fundamental concepts of P4 programming language along with the architecture and targets to which a solution may be deployed. It then covers some basic cybersecurity concepts, focusing on port scan as the subject of detection. To address this subject, the chapter further explores various notions, algorithms and structures related to intrusion detection, particularly anomaly detection. Chapter 3 introduces P4-CAVODE, the proposed solution for network anomaly detection. It also addresses considerations regarding how it is optimized to take advantage of the hardware capabilities. In order to evaluate this solution, Chapter 4 contains the description of the tests performed on different

scenarios, as well as their respective results and analysis, which focuses on key factors expected from the detection system and on how it achieves the objectives outlined in Section 1.2. Lastly, Chapter 5 provides a summary derived from the evaluation of the developed solution and describes potential improvements and future directions for its further development.

Chapter 2
Related Work

This chapter provides an overview of fundamental P4 programming concepts, along with an examination of the architectures and targets pertinent to this research. It then transitions into the topic of cybersecurity, offering insights into port scanning and mechanisms capable of detecting it. Additionally, it explores some practical lightweight data structures and techniques that detect deviations in data streams, which are useful in the ambit of anomaly detection. This foundational knowledge sets the stage for understanding the developed solution and its components discussed in the subsequent chapter.

2.1 Programming Protocol-independent Packet Processors (P4)

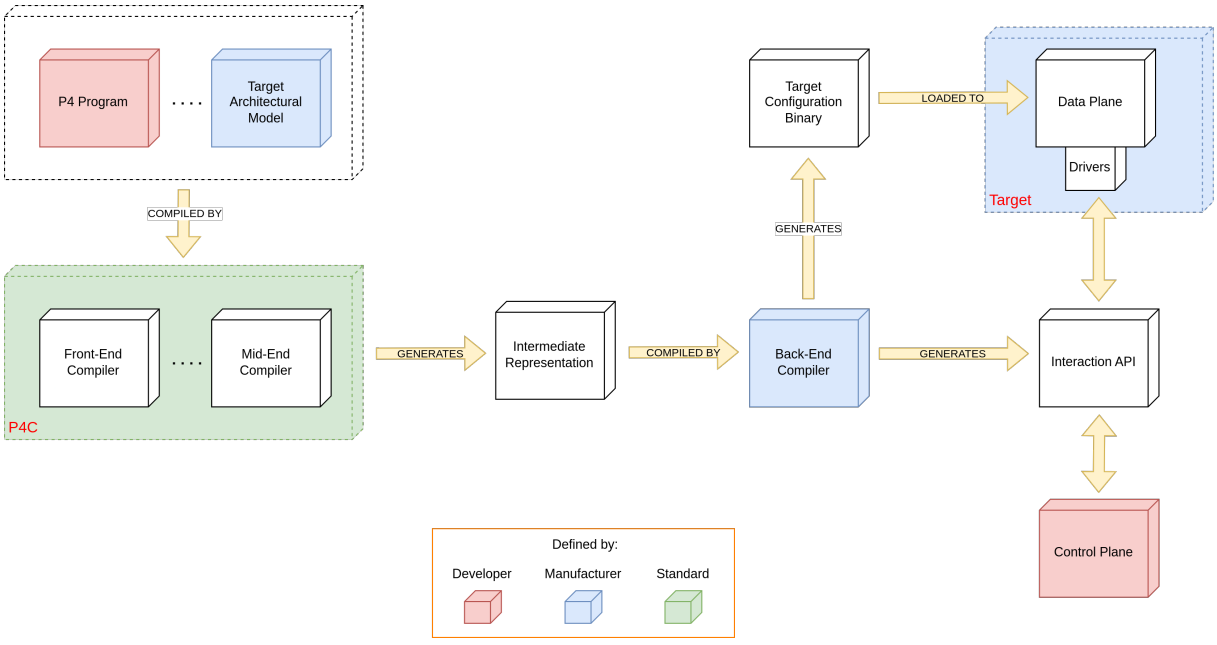


Figure 2.1: Compilation workflow [68, 91, 94]

As mentioned in Chapter 1, P4 is a highly expressive language suitable for defining configurations on how packets should be processed by the data plane in network equipment. In a P4 programmable network device, no data plane functionalities or network protocols are defined in advance. Rather, they need to be

implemented via a developer-defined P4 program that must comply with the architecture provided by a target device manufacturer. Figure 2.1 illustrates the compilation process of a P4 program until the point it is loaded into a data plane of a target hardware or software device. It starts by passing the program to the P4 language's standard compiler, P4C [92]. P4C is composed of a front-end that handles parsing, syntaxes and target-independent semantics, and a mid-end that performs transformations and optimizations. The resulting product is an Intermediate Representation (IR) that details the program in a format meant to be interpreted by a back-end compiler, which is also provided by the target manufacturer [72]. This compiler ensures that the program adheres to the constraints of the architectural model and, if so, it produces a binary that is loaded into the data plane. Additionally, it establishes an Application Programming Interface (API) that is used to communicate between the control plane and data plane through the drivers associated with the target [41].

As stated in Section 1.1, the prototype of the anomaly detection system developed for this dissertation should be capable of running on both software and hardware targets. Target devices vary significantly according to their purpose. In general, the most common associations for them are Application-Specific Integrated Circuits (ASIC), software or Central Processing Unit (CPU), Field Programmable Gate Arrays (FPGA), and Network Processing Unit (NPU).

ASIC, as the name suggests, are highly specialized targets aiming to achieve high performance, low latency and low jitter. They do so by highly restricting the expressiveness in the instruction set and constructs in the P4 language [70]. For instance, the other types of targets mentioned usually allow developers to define and invoke external operations or functions not natively supported by P4, whereas ASICs do not [61].

One notable example is the Intel Tofino [82, 83], a series of ethernet switches, which boast the highest throughput among the existing target devices. Through the mentioned restrictions, advanced pipelining and parallelism, each of its ports is capable of processing hundreds of Gigabits per second, up to a total of 12.8 Terabits per second (Tbps), depending on the hardware model. The Tofino switch features multiple programmable pipeline stages that operate in parallel, each composed of multiple Match Action Units (MAU) that allow for the storage of stateful objects [55]. Each pipe operates independently, allowing them to run distinct programs if desired. Additionally, pipes can be interconnected to create longer processing pipelines, enabling more complex programs [70]. With its high performance and specific restrictions, it is recommended for hyperscale environments, data centers, cloud services, and service provider environments. In 2023, Intel announced it would cease supporting the development of the next generation of these devices[1].

[1] https://groups.google.com/a/lists.p4.org/g/p4-dev/c/6TYK_aNWvQk

Software targets essentially consist of programs designed to emulate the behavior of hardware devices, running as software on generic CPUs. These targets are not limited by the same restrictions as specialized hardware, making them more flexible by enabling the development of custom programs and architectures. As a consequence of the lack of such restrictions, the offered performance is also significantly lower than hardware targets [65, 70]. The most popular P4 software target is the BMv2 [71], which is also the reference open-source software switch in the P4 language, represented by the *simple_switch* and *simple_switch_grpc* targets. As such, it is ideal for educational, development, or debugging purposes under specific scenarios. It only supports a single thread to manage ingress pipeline, whereas the egress pipeline is handled by multiple threads and therefore, it can only achieve a maximum throughput of 1 Gigabits per second (Gbps).

FPGA are integrated circuits focused on providing flexible architectures through the usage of Hardware Description Languages (HDL) [70]. These languages enable the definition and calling of custom external functions and operations not inherent to P4, but mastering them can be time-consuming [44]. The NetFPGA Sume board [45] is a high-performance reconfigurable network platform that incorporates an FPGA and primarily serves as an experimental tool for research and prototype development purposes in the networking field [64, 70]. It features four high-speed interfaces, each capable of reaching 10 Gbps and can function as IP router, switch or even as a Network Interface Card (NIC) [30, 44]. Despite that, it does not support every functionality of a standard switch, as well as some native P4 features [44, 61]. For example, metering functionalities and certain types of matching rules in the match-action pipeline are not supported [57, 58]. According to the NetFPGA organization[2], the NetFPGA SUME board is supplied by Digilent. However, as noted on Digilent's store page[3], this target device has been retired and is no longer available for purchase.

Similarly to ASIC, NPU are specialized hardware, but to a lesser degree, which allows the developed programs to be more flexible and complex. Its architecture enables high throughput and low latency by using multithreaded cores that distribute packets efficiently [70]. An example is the Netronome's Network Flow Processor (NFP), which represents a family of network flow processors designed for high-performance flow processing and throughput, primarily programmed through languages like C and P4. The NFP-4000 [73, 88] is designed to handle high packet processing rates using advanced parallel processing and multithreading techniques. It features 60 processors called Flow Processing Cores (FPC), specifically dedicated to packet and flow processing. Each FPC can manage 8 threads, allowing the SmartNIC to process up to 480 packets simultaneously. Additionally, it includes hardware accelerators that relieve the

[2]https://netfpga.org/NetFPGA-SUME.html

[3]https://digilent.com/reference/programmable-logic/netfpga-sume/start

FPC from routine processing tasks, such as load balancing, packet modifications, and traffic management, among others, further enhancing performance and efficiency [39]. These features make it ideal for tasks where performance and efficiency are critical, such as cybersecurity.

In the end, the chosen target for the software is BMv2 due to its documentation. Regarding the hardware target, the support for developing next-generation Intel Tofino ASICs has been discontinued, and the NetFPGA SUME is no longer available for purchase. For those reasons, the hardware concerning this dissertation is the Netronome NFP-4000 Agilio CX 2x10GbE Smart Network Interface Card (SmartNIC). While both chosen targets are involved in the processing of network packets and can be programmed using P4, their underlying characteristics reflect their intended use cases: BMv2 for accessible, flexible and controlled environment for the development of proof-of-concept implementations, and Netronome's NFP-4000 for high-performance, low latency and efficient packet processing in real-world networking environments.

2.2 Target architecture - V1Model

The targets concerning this dissertation are Behavioral Model v2 (BMv2)'s simple_switch [71], and Netronome's NFP [73, 88], which both adopt the same core architecture, the V1Model [98].

Figure 2.2: V1Model forwarding pipeline [25]

The programmable structure pipeline of this architecture is depicted in Figure 2.2. The shown components, except for the *Traffic Manager*, are labeled as programmable blocks and are tasked with defining the processing of each packet at different stages through sets of instructions. During this packet processing, intermediate data known as metadata is generated. This data is shared and modifiable between all programmable blocks and mainly consists of two types: User-defined and Intrinsic. The former, as the name suggests, relates to custom structures defined by the developer to store and modify data relevant for the program's needs. For instance, network protocols can be declared as part of this metadata so their parameters can be extracted and used by the different blocks. Additionally, some operations in a block may be dependent on variables set or modified in a prior block. As for the intrinsic structures, they are given by the architecture and include information regarding ingress and egress ports, packet priority, multicast group, queue depths, etc [98]. The modification of metadata, packet header fields and other structures is typically performed by actions, which essentially follow the concept of functions. Both actions

and blocks can be declared with three types of argument according to their intended usage: *in* for input (read-only), *out* for output (write-only), and *inout* for input and output (read-and-write).

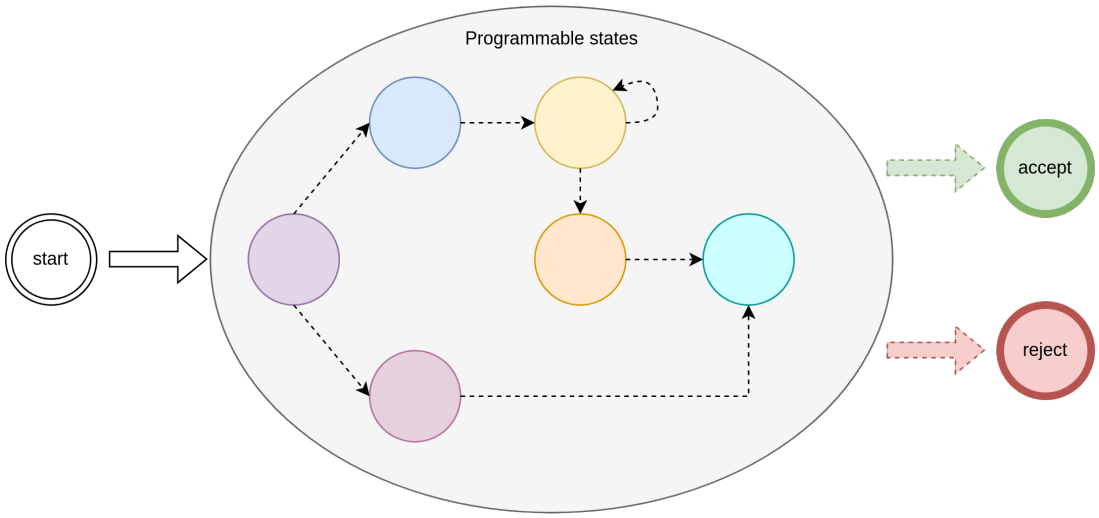


Figure 2.3: States of the parser [43]

Regarding the first block, the *Parser*, it is fundamental to establish, beforehand, the custom metadata and the appropriate headers for any desired protocols. This information will then be extracted from each packet by mapping them based on a state machine notation, in which, as can be seen in Figure 2.3, there are already 3 predefined states [43]. The initial state is always *start* and from there, it progresses to intermediate states where the information is extracted and if desired, used to match and transition to other intermediate states. At the discretion of the intended program, multiple intermediate states can be defined, allowing further transitions among them. However, the final state of a packet must always be either *accept* or *reject*, indicating that the parsing process was either successful or not, respectively [60].

The next control block, known as *Ingress Processing*, is responsible for applying checksum verifications and defining how the extracted data should be processed upon its arrival. This block typically involves operations that perform table lookups and actions that may modify other stateful structures or packet headers. During these operations, decisions are made regarding its forwarding destination or whether the packet should be dropped or replicated.

Tables, also known as *match-action tables* are one of the two types of stateful structures in P4 and are used to carry out the forwarding process [68]. They require a matching criterion type and at least one key to match their entries. This criterion can be *ternary*, which uses a mask to specify which sets of consecutive bits are compared, *exact*, which requires that all bits in the keys match, and *Longest Prefix Match (LPM)*, which chooses the key with the greatest number of consecutive matching bits. The program does not handle adding entries to the table, that responsibility falls to the control plane. Besides that, it

is necessary to specify which actions should be applied if there is no match and which actions should be applied in the event of a match occuring within the table. The other type of stateful structures are called *externs* (e.g., registers, counters), which can be read and written by both control and data plane [68]. This makes them particularly useful when the intent concerns implementations within the data plane since there is a facilitated maintenance and modification of states across various packet instances without necessarily requiring control plane intervention. Consequently, certain tasks can be offloaded from the control plane to the data plane, enhancing overall efficiency. Additionally, various statistics can be monitored and the system can also take actions regarding network events influenced by these statistics. When declaring stateful structures, it is necessary to specify either the number of positions or, depending on the structure, both the number of positions and the bits per position used. The memory for these declarations is usually allocated to on-chip resources, specifically Static Random-Access Memory (SRAM) or Ternary Content-Addressable Memory (TCAM) [25, 51].

The *Traffic Manager* is responsible for handling how packets are queued and buffered for transmission. Generally, its functions and implementation are predefined by the target device, which makes them inherently fixed and therefore, not programmable via P4. However, some targets, such as Intel Tofino offer a certain degree of customization regarding this component, namely in class-of-service queues, scheduling policies, congestion management, etc [66, 78]. Simultaneously, recent advancements are pushing towards making it more programmable [58].

The *Egress Processing* block is tasked with updating checksums and the final processing of packets before they are transmitted out of the device. Similarly to the *Ingress Processing* block, table lookups and actions can be defined. An example of this final processing is when multicast is considered. In this case, the block handles packet replication by ensuring that the ingress port is not the same as the egress port so it can be forwarded to multiple destinations [25]. Lastly, the *Deparser* block is responsible for assembling the egress packet. It does so by reconstructing the packet using the previously modified headers, after which the packet is sent to the next destination [60].

2.2.1 Limitations, challenges and differences between targets

P4, while flexible and efficient for data plane operations, presents several limitations regarding its constructs that can pose challenges during the development of a P4 program. Some of those are that P4 does not include support for [59, 68]:

- Floating-point arithmetic and functions such as logarithmic calculations;
- Loops and recursive functions, since the processing of packets is linear;

- Pointers, references, or dynamic memory allocation;
- Built-in mechanisms for handling exceptions or errors that may occur during packet processing.

These limitations enforce strict constraints on the features and constructs of the language, which in turn allows it to be more efficient and have better performance at the data plane level. Besides, with these constraints, constructs are less complex, granting better portability across different targets.

Regarding target specifics, both the chosen software and hardware targets share the same core architecture, which is convenient in terms of the portability process since most constructs will be similar. Despite that, the hardware component still requires several key aspects to be taken under consideration [58]:

- In the Netronome, the registers are not synchronized with the in-hardware flow cache, which, along with cached lookups, could lead to unpredictable outcomes when accessing registers that influence control flow. This can be addressed by disabling the cache-flow option on the SmartNIC, which could impact the algorithm's performance;
- The multi-threading processing on the ingress results in the desynchronization of registers' read and write operations;
- BMv2 employs a 48-bit format to express timestamps in microseconds, whereas the SmartNIC does it with 64-bits, using two distinct 32-bit fields, one for seconds and another for nanoseconds;
- Netronome places its queues after the egress pipeline, which results in the absence of some standard metadata fields that typically indicate queue occupancy. Since both the ingress and egress pipelines precede the actual queuing and scheduling functions, the egress processing lacks visibility into actual queuing delays.

2.3 Port scan

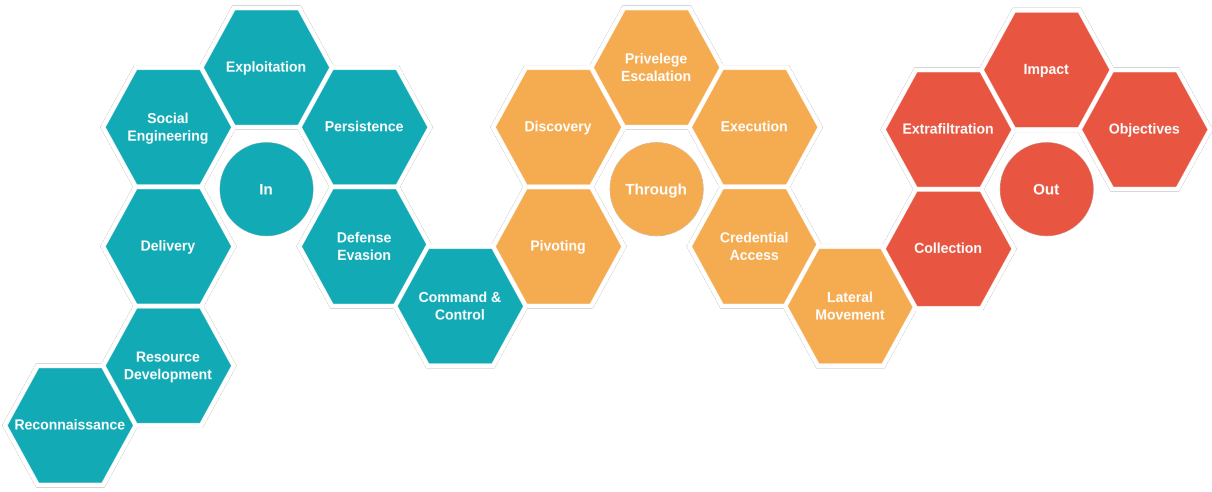


Figure 2.4: Unified Kill Chain stages [93]

In the domain of cybersecurity, there are multiple existing ways of describing what composes the process of an attack. For instance, Figure 2.4 describes the Unified Kill Chain [93], a recently proposed model that not only encompasses standardized frameworks and models such as Cyber Kill Chain [69] and MITRE's ATTCK [87], but also extends them by integrating additional phases and insights tailored to the dynamic nature of cyberattacks.

Typically, what all these types of models have in common is that when there is an attack, it is tied to an initial stage which is necessary to gather information about its targets. This procedure is known as *reconnaissance* and can be classified in two ways, passive and active. The passive approach attempts to minimize contact with the target by acquiring information through public sources (e.g., websites, social networks, public databases, job offers, browsing, etc), while the active approach involves a greater risk of detection to the attacker, since it requires more direct contact with the victim, typically by sending custom packets for the information that one wishes to acquire, and consequently analyzing the response obtained [23, 62]. Although it poses a higher risk, it also allows for a more select set of data to be obtained, such as the range of IP addresses used, active machines, open ports, access points, operating system type and version, services, network mapping, and more [15].

One of the main techniques used in active *reconnaissance* is port scanning. This consists of identifying the state of ports at the transport layer of the TCP/IP model. The objective behind it is to discover services or applications running on open ports that could possibly constitute a vulnerability for the system. It does so by sending packets to various ports of a target and analyzing the responses to determine which state the associated ports are into [6].

To better understand the purpose of port scanning, it is essential to have a notion of the different states at which ports can be. The most common designations, and the ones this dissertation will consider are as follows [49]:

- Open → Ports in this state are actively listening for incoming connection requests. It indicates that a service or application is running on the port and ready to accept network traffic, where an application is actively listening for incoming requests;
- Closed → The port is accessible but no service or application is waiting to communicate. This means that depending on the protocol, it might still be possible to know if the target is active;
- Filtered → In this state, incoming packets are filtered, ignored or blocked, so there is typically an association with the use of firewalls or other security methods. Usually, there is no response from these ports, making it challenging for attackers to actually know if the port is open or closed.

An open port itself is not inherently risky. The true risk lies in the service that is running on that open port. In a sense, one could describe an open port as a potential pathway for a potential attack [9]. The configuration of a service and its lack of security updates for known vulnerabilities can lead to exposures that attackers can exploit to compromise the system. Therefore, when a hypothetical attacker identifies a port as open, the real danger arises from the way in which they can interact with the associated service.

Although it can be used in preparation for an attack, it does not necessarily imply that this technique is malicious. A security manager or other similar entity can also put it into practice to prevent potential exposures by determining which ports are open on its system and closing them if they are deemed unnecessary [29].

2.3.1 Existing ports and their classification

Understanding existing classifications of the port identifiers may help identify which services, programs or protocols typically operate on these ports. This knowledge not only enables a better comprehension of their traffic behavior but also facilitates the assessment of any vulnerabilities associated with them.

Internet Assigned Numbers Authority (IANA) [84] is a standard authority responsible for officially allocating port numbers for specific purposes. It uses a 16-bit namespace which results in 65 536 possible ports [20], which are subdivided according to the following classifications for their respective ranges [34]:

- Well Known or System ports → Range from 0 to 1023

  They are used in standardized essential services.

- User or Registered ports → Range from 1024 to 49151

  These ports are used for less critical services compared to well-known ports and require authorization from IANA.

- Dynamic, Private or Ephemeral ports → Range from 49152 to 65535

  They are not designed for any specific service and can be used for any purpose. As such, they are typically used for customized services or temporary connections such as client-side communications [20].

The known and the registered ports are in one of 3 possible states [20]:

- Assigned → It is assigned to the service indicated in the IANA registry;
- Unassigned → It is available for assignment via a request;
- Reserved → It is not available for regular allocation, but is associated with IANA.

2.3.2 Characterization of port scan strategies

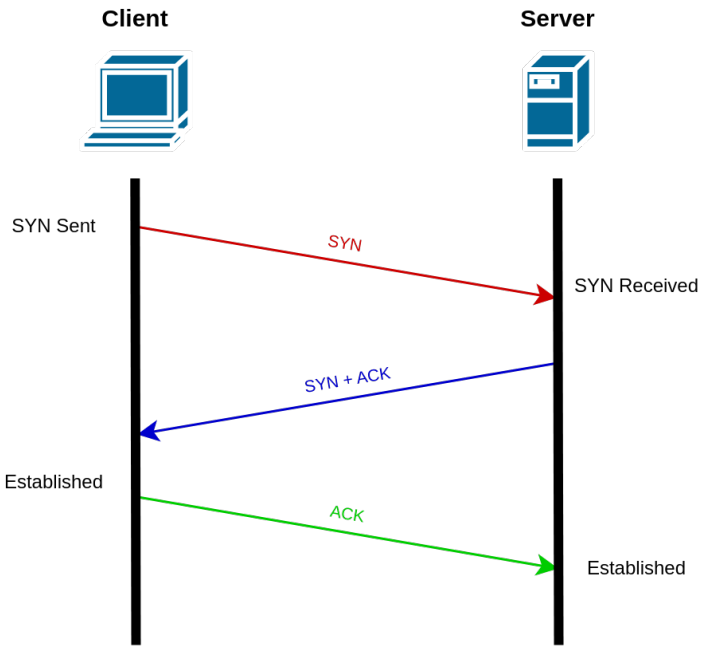


Figure 2.5: Three-way handshake [32]

Since port scanning pertains to the transport layer of the TCP/IP, its request strategies are based on the UDP and TCP protocols. In the case of communication with this second protocol, the three-way handshake

protocol is used to establish a connection. As shown in Figure 2.5, it does so by a client communicating its intention to establish a connection with a server. The server acknowledges those intentions and agrees to the connection. Lastly, the client confirms the receipt of the server response. This process ensures both sides are synchronized and ready to communicate.

While ultimately port scanning has its objective set in finding out which ports are open, there are several ways of classifying them according to the way they act or are built. Some scans are optimized for speed, others attempt to evade detection by security systems, and some exploit flaws in protocol descriptions to acquire the desired information. The classification considered in this dissertation takes into account the headers used, the multiplicity of the source or target, and the timeframe over which the scan occurs.

Headers

Most relevant strategies that rely on header modifications involve the flags associated with the TCP protocol, hence, having a general idea of how these flags work is important [3]. These flags are as follows:

- `SYN` → As in Figure 2.5, it is typically used to establish a connection and to synchronize the packets' sequence numbers with the receiver;
- `ACK` → Used by the receiver to signal that a packet has been successfully received;
- `FIN` → Indicates that there are no more packets to be transmitted by the sender, in other words, the packet with this flag is typically the last one;
- `RST` → Used to reject or abort a connection. It is generally used when a packet is invalid or unexpected;
- `PSH` → On a connection, when packets are sent, they are stored in a transport layer buffer before being processed. When this flag is used, all the data stored in the buffer begins to be processed;
- `URG` → Sending a packet with this flag indicates that this packet is urgent and should be processed before the other packets in the buffer.

With this in mind, below, are some of the most well-known port scanning strategies based on their headers [75]:

- TCP `SYN` or Half-Open Scan → In this type of scan a `SYN` packet is sent and, if the response consists of a `SYN ACK` it means the port is open and, if a `RST` is received, it is closed. The lack of response or the reception of a ICMP error could indicate that the port may be filtered [26];

- TCP Connect Scan → It closely resembles TCP `SYN`, with the key difference being its use of the *connect()* system function call, resulting in the completion of a three-way handshake, making it a longer process. If the connection is established, the port is open and the connection is terminated, otherwise it is closed. The filtered condition is determined in the same way as the `SYN` scan;

- TCP Window Scan → In the TCP header there is a field called window size and the value of this field indicates the number of bytes it is willing to receive in the next packet. This strategy involves sending a packet with the `ACK` flag and if the reply is a `RST` with positive window size, it means that the port is willing to accept another packet, meaning it is open, and if it is 0 the port is closed;

- TCP `FIN, NULL, XMAS` and `Maimon` Scans → These strategies exploit specific descriptions in Request For Comments (RFC) 793 [3]. According to this RFC, if an open port receives a packet without the `SYN`, `ACK`, or `RST` bits set, the packet should be dropped. If the port is closed, then it should answer with an `RST` packet. As such, the `FIN` scan sends a packet with the `FIN` flag, `NULL` does not contain any flags and `XMAS` uses a combination of `FIN`, `PSH` and `URG`. This scan method only works if the target system fully employs the RFC, which most systems that take port scan into account do not since these are widely known port scan strategies. Additionally, if the port is filtered, it might also drop the packet, making it inconclusive if the port is open or filtered;

- `Maimon` Scan → This technique works similarly to the `FIN`, `NULL` and `XMAS` scans. Some systems derived from Berkeley Software Distribution (BSD) do not entirely follow the part of the RFC 793 that states that an open port receiving packets without `SYN`, `ACK` or `RST` bits set will respond with a packet drop. Instead, if a packet with the `FIN` and `ACK` flags is sent, it causes that same effect. The remaining details are identical to the `FIN, NULL, XMAS` scans;

- UDP Scan → The idea is to send an UDP packet and wait for a reply. If this response is "ICMP Port Unreachable" then the port is closed. If no response is received then the port is open or filtered, and if a response other than the one mentioned previously is received then it is open. In most cases, the packet sent has no payload. However, for certain well-known ports, concerning specific protocols, sending a packet with a specific payload may help acquire information on the state of the port;

- TCP FTP Bounce Scan → This type of scan is based on the File Transfer Protocol (FTP). Essentially, a user connects to an FTP server and communicates their intention to send a file, using a `PORT` command along with the target IP address and port. The server, in turn, communicates this

intention to the target. If the target port is open, the server tells the sender that the transfer has been successfully completed. Otherwise, it indicates that the connection could not be established.

Since it uses an intermediary to carry out the scan, it is a slower strategy than the others, which can make its detection difficult. Considering its characteristics, there is limited availability in terms of FTP servers and most of the existing ones do not allow third-parties to access this sending functionality, as it may pose a risk to the security of a system [26];

- TCP Fragmentation Scan → This strategy usually employs one of the TCP `SYN` or `FIN` scans. Instead of sending an entire packet, it divides it into small fragments, which are marked with the *more fragment* bit set to one and, when the last fragment arrives, the said bit is set to zero. The fragments are then assembled together in order to reconstruct the original packet. The idea behind this strategy is that if the fragments are small enough, they may split the TCP header across multiple fragments, allowing them to evade detection systems that do not perform packet reassembly before analysis [26, 74, 80].

Table 2.1 summarizes the described header strategies in terms of their protocol, three-way handshake completion and flags used [75].

Table 2.1: Port scanning header strategies [75]

Strategy	Protocol	Three-way handshake completed	Used flags
TCP SYN Scan	TCP	No	SYN
TCP Connect Scan	TCP	Yes	SYN
TCP Window Scan	TCP	No	ACK
TCP FIN Scan	TCP	No	FIN
TCP NULL Scan	TCP	No	–
TCP XMAS Scan	TCP	No	FIN, PSH and URG
UDP Scan	UDP	–	–
TCP Maimon Scan	TCP	No	FIN and ACK
TCP FTP Bounce Scan [26]	TCP and FTP	Yes	–
TCP Fragmentation Scan [26]	TCP	No	SYN or FIN

Multiplicity

Regarding the multiplicity factor, whenever it concerns the source of the scan, it can either be a single-source scan or a distributed scan. The former, as its name suggests, is performed by a single source IP address. Thanks to this, detection systems that keep track of IP addresses will easily identify the suspect

whenever a port scan is detected. As for distributed scans, they have multiple sources IPs, which not only make it harder to pinpoint the actual responsible for doing it, but it may also be more difficult to actually identify as a scan since it is necessary to correlate the different packets properly, which detection systems based on flows with the same IP addresses may not be able to do [17, 19, 21].

When multiplicity pertains to the target there is also [21, 26]:

- Vertical Scan → Scan some or all ports of a target.
- Horizontal Scan → Scan a specific port of multiple targets.
- Strobe Scan → Scan multiple ports of multiple targets.
- Block Scan → Scan every port of multiple targets.

With the exception of vertical scan, all these strategies concern multiple targets, which may present a challenge in terms of detection to IDS that do not consider different destination IP addresses, since they may not be able to make an association between the generated probes.

Timeframe

According to the length of time during which the scan takes place or the interval in between each probe, there are the designations *fast port scan* and *slow port scan* [50]. In a *fast scan*, the delay between probes is very low, allowing multiple requests to different ports to be sent in a relatively short time window. These requests typically range from tens to thousands per second. Depending on how a detection system is defined, a burst regarding the number of different ports probed is expected, making it easier to identify.

Slow scans are executed over an extended time window, with their requests being performed every few seconds or longer [50]. They can be more difficult to distinguish from a legitimate request, since in a network there is a high volume of traffic and keeping relevant data for analysis would be very costly in terms of resources, thus the state of the packets is only kept for a limited time, which complicates their association with previous packets. If the time interval between the transmission of these packets is longer than the time for which the state of the packets is kept in the system, detecting the occurrence of scanning could prove to be a major challenge [42].

Other designations

A commonly used term is stealth scan, which focuses on avoiding detection by security systems. This type of scan is not necessarily tied to a single strategy and, therefore, it may encompass the combination of multiple strategies explored during this section. Some of these are:

- Not completing the three-way handshake protocol as a way to generate fewer logs, as is the case of the aforementioned strategies in Table 2.1;
- Distributing the probes across multiple senders, so that a direct correlation can not be made through its sources, as in a distributed port scan;
- Expanding the timeframe of the scan so that correlating packets becomes harder, as in a slow port scan.

In counterpart, the absence of these characteristics is often addressed as brute-force port scan, in which the scan is carried in a more aggressive manner without trying to hide any of its activity from a detection system [15, 26], hence its name. Therefore, it usually consists of a single-source fast scan that completes the three-way handshake.

2.4 Technologies against intrusion

While port scanning itself is not an act of intrusion, the underlying idea of detecting both port scans and intrusions shares many similarities. Both activities involve monitoring events and identifying suspicious activity, with real-time detection being ideal, so that it may be responded to. As such, exploring intrusion protection mechanisms that can be adapted to the characteristics of port scanning may effectively address its detection.

Typically, intrusion protection falls into two categories, detection, under the form of Intrusion Detection System (IDS) and prevention, under the form of Intrusion Prevention System (IPS). Detection is essentially the process of observing and analyzing events on a machine or network. This analysis tries to distinguish these events based on whether they are suspicious or not. If the former is the case, the IDS notifies an administrator or someone equivalent. In turn, an IPS works in the same way, thus including the detection component, however, there is an important difference, which is in terms of the action to be taken if an event is determined to be malicious. This action may involve blocking the event and, depending on how it is configured, the source of the event. In this sense, it can be said that an IDS takes a passive role while an IPS takes an active role [13]. As for the detection component mentioned above, it can be classified depending on the type of events it observes. The most common classification is [13, 40, 63]:

- Network Intrusion Detection System (NIDS) $\rightarrow$ It observes the traffic of one or more networks and analyzes the protocols used to determine which of this traffic could be harmful [13]. Although it is able to detect attacks as they occur, it is unable to indicate whether any of them were successful or not [40].

- Host Intrusion Detection System (HIDS) → It monitors the activity of a single host. The activity monitored is based on the device it is running on, and can consist of both wired and wireless network traffic, system and audit logs, processes, file access and modification, and system and application configuration changes [13]. It is also possible for an HIDS to have agents installed on various hosts, where each agent monitors specific components of that host. These agents then send the collected data to central servers. These servers store the data in a database and, upon detecting any discrepancies, take appropriate action [10, 63].

2.5 Detection mechanisms

Identifying the occurrence of an intrusion requires mechanisms capable of discerning legitimate activities from malicious ones within a network or system. Each of these mechanisms offers unique advantages and operates under different assumptions, making them suitable for different scenarios without dismissing the possibility of being integrated together. This section explores some of the most commonly used detection mechanisms and provides practical examples to illustrate their application.

2.5.1 Threshold detection

It is the most basic method and classifies an event as suspicious if its occurrence exceeds a predefined threshold. This threshold tends to be difficult to determine since data patterns can vary widely, increasing the risk of either overlooking genuily suspicious events or incorrectly flagging legitimate activities. Thus, this technique alone is usually considered inefficient and should be used in conjunction with others [5].

The implementation by Dabbagh et al. [21] is dedicated to port scans, specifically *slow scans*, and, consists of storing relevant packet data within small time windows to reduce memory costs. Essentially, there is a time window where the number of connections made by each source IP is stored and, if a certain threshold is surpassed, then this IP can be considered suspicious and thus kept on a suspect list for a longer time window than the previous one. If in a subsequent time window it is considered suspicious again and it is still on the suspect list, a notification is sent to a firewall or administrator. More specifically, this method classifies as suspicious based on the assumption that a legitimate user will not try to connect several times to closed ports, and takes into account the most common scanning strategies, TCP Connect Scan, TCP SYN Scan and `FIN` Scan. For the first case, the difference between the number of `SYN` packets received from an IP and the number of `SYN/ACK` packets sent by the server to it (which is only sent if it is open) is calculated. In the second case, it calculates the number of `SYN/ACK`s sent by the server to the IP source, that were not responded with a subsequent `ACK` by the corresponding IP, indicating that

the connection was terminated before the completion of the three-way handshake. As for the last case, it calculates the difference between the number of `FIN` packets received from an IP and the number of `FIN` packets sent by the server to that IP (which is only sent if it is open). In essence, what is being done in each of these calculations is to see which IP addresses are initiating communication with a closed port and classifying the associated users based on the result of those calculations. If the result of any is equal to 0 then it is considered a legitimate user, if it is equal to 1 it is added to the list of suspects and if it is greater than 1 it is an attacker. The primary issue with this implementation lies in properly adjusting the time window thresholds so that the stored data for each IP does not overwhelm the system. Additionally, the detection is limited not only to the mentioned flag strategies but also to distributed scanning since its associated IPs could be labeled as suspicious and perhaps never as attackers due to not exceeding the threshold.

Singh [17] proposes a model for detecting distributed port scans with the assumption of TCP connections in which terminating the connection in the handshake process is considered abnormal behavior. To determine whether port scanning is taking place, a structure called the Partial Completion Filter (PCF) [7] is used. It consists of a set of stages running in parallel, each of which uses hash buckets with counters as a structure. To determine the appropriate hash bucket for each flow, a hash is computed based on the combination of the source IP address and the destination port, which act as keys. When a termination of a connection is detected, the associated counter is incremented, otherwise, it is decremented. If all the counters of a key, across the different stages, surpass a defined threshold, this key is seen as suspicious [7].

2.5.2 Rule-based detection

It is a method that evaluates data according to a set of predefined conditions that if triggered may deem an event as suspicious and therefore should be analyzed [5]. Kanlaysiri et al. [5] developed a model focused on port scan detection. This model consists of two main phases, entitled *Feature Selector* and *Decision Engine*. In the former, the most essential packet parameters are chosen, those being the source IP address, TCP flags, the number of received packets, the destination port number, and time intervals between consecutive packets. These parameters are then used to generate a 5-tuple. In the latter phase, it is decided what may constitute an intrusion, and an analysis of this 5-tuple is carried out. If there are no flags or if any of the `SYN`, `FIN`, and `FIN/PSH/URG` flags are present (these flags define some of the request strategies investigated in Section 2.3.2), then the time interval between consecutive packets connecting to different ports is compared among packets with the same source of origin. The system assumes that,

under normal circumstances, a source address should not send multiple packets to different ports of a host within a very short time frame, such as one second. Therefore, if more than one packet is sent to different ports within this interval, and this behavior repeats more than a specified number of times, the source address is flagged as suspicious. This model seems to be successful when it comes to some of the previous strategies, but it has two major problems. The first is that the decision layer works mostly on threshold detection and the thresholds in question may be difficult to adjust for services of different dimensions, in addition to the fact that the time interval used may exclude *slow scans*. The second is that it only works for single-source port scans, thereby excluding distributed ones.

2.5.3 Misuse detection

Misuse detection, otherwise known as signature-based detection, is a simple mechanism that operates by comparing provided data against a database with signatures or patterns of known vulnerabilities or flaws in a system. If there is a match in this database, then the data is classified as suspicious. This feature is its greatest strength as it gives it a high accuracy in detecting attacks, however, if they are new or unknown it is completely futile. The main challenge with this type of mechanism is creating a signature capable of including all variants of an attack without excluding legitimate activity, which is why it usually acts as a subcomponent in a broader detection system [5, 10]. An example is Snort [95], an open-source packet sniffer, packet logger, and IPS focused on network monitoring, which relies on both rule detection and misuse detection to match and identify suspicious activity. It maintains a database of rules and signatures pertaining to known malicious or suspicious traffic, allowing users to add custom entries and specify appropriate responses for each.

2.5.4 Model-based detection

The aim of this type of detection is to create models capable of defining behaviors to be compared with the traffic taking place on a network. The model can exemplify both normal and malicious behavior and, unlike misuse detection, it can adapt to new attacks by making changes to the model (e.g., using other detection mechanisms to add new patterns to the model). The main challenge with this type of intrusion detection is that building a model that is sufficiently representative and accurate in relation to the desired traffic is an expensive task in terms of both cost and time [5, 12].

Kim et al. [46] utilized the `CSE-CIC-IDS2018` dataset [81], which comprises various attack formats, to develop a Deep Learning (DL) model. This approach employs convolutional neural networks [31], transforming the dataset into an image to generate a basis for identifying malicious traces. Incoming

traffic is also converted and then compared to this basis to determine whether it is benign or malicious. The major concerns with this method include the possibility of overfitting and the potential lack of quality in the attack formats covered by the dataset.

2.5.5 Anomaly detection

Anomaly detection refers to the separation of normal from irregular behavior in a series of data. In the context of network security, it generally uses statistical models or techniques and sets of rules to quantify and qualify acceptable behaviors and patterns. If an event falls outside the thresholds acquired during these calculations, then it is marked as suspicious [5]. Although it is an efficient mechanism for detecting unknown attacks, it is also prone to generating false alarms. These can be false positives, where normal traffic is classified as anomalous, or false negatives, where anomalous traffic are classified as normal [63].

Ananin et al. [35] propose a solution focused on port scans, using a mathematical model to determine anomalies. In this model, a vector is created for each received packet, containing the source IP address and the TCP flags. These vectors are then stored in a table for a defined time interval. In this table, the anomaly index is calculated based on the frequency of each specific flag combination in each vector stored. Initially, the network's normal characteristics are defined and taken as a reference for future traffic comparison. To do this, the anomaly index of these characteristics is calculated and a variation value is defined, forming a range of values. For all future traffic, if it is contained within the limits of the range it is considered legitimate traffic, otherwise it is considered malicious. First of all, this technique heavily relies on the quality of the initially captured traffic. If this traffic is not representative enough, there is a possibility that future normal traffic might not fall within the anomaly index's range of values. Additionally, it focuses on detecting based on the flags used and the source IP. Therefore, with an adequate reference traffic, it may be possible to cover strategies that consist of header modifications such as flags, but perhaps not on other fields like other protocols or window size, as explored in Section 2.3.2. Regarding the source IP address part, if the detection system only concerns the protection of a single IP address, it might be efficient against distributed scans. However, if the initial traffic does not encompass every possible target, a distributed scan involving multiple targets might struggle to trigger the alarm effectively. Additionally, the analysis is performed under time windows that may be too short to identify as suspicious the occurrence of *slow port scans*.

In anomaly detection, it is normal to have structures that keep track of data for given periods. However, depending on how long those periods last, storing the entirety of features extracted from packet flows generally leaves a large footprint in terms of storage. To deal with this issue, computing those features

with hash functions and mapping them to positions of a data structure is often a practical idea. These are known as membership structures, and, at each position, they store a bit indicating whether the corresponding combination of features is already present. Furthermore, this detection method is frequently combined with change detection techniques applied to data series, which typically rely on a statistical measure as a parameter that triggers an anomaly when a deviation is detected. This parameter is usually associated with features extracted from the packet flows, requiring a data structure that links the flows to their corresponding parameters. This structure is referred to as a counting structure and works in a similar way to membership structures, in the sense that the flows are mapped to a position in the structure and, instead of having a bit that sets their presence, they may have a counter or other additional data.

2.6 Data structures used in anomaly detection

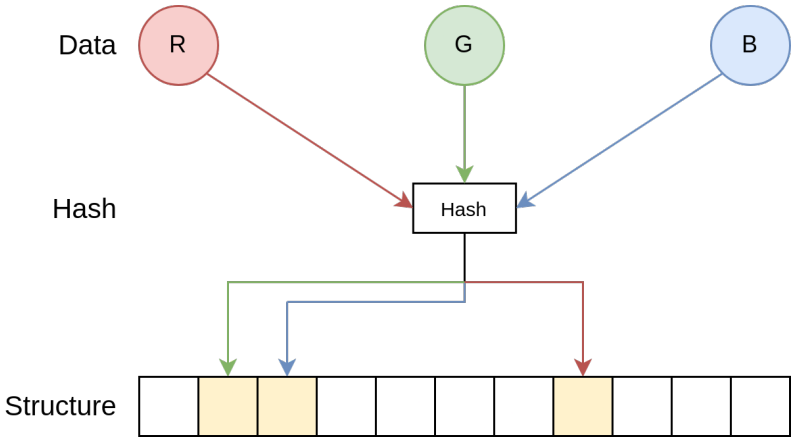


Figure 2.6: Generic membership structure

As mentioned in Section 2.5.5, anomaly-based intrusion detection focuses on identifying deviations from expected or normal behavior within a network or host. To achieve this, it leverages a range of mathematical and statistical models, with one of the most crucial aspects being the concept of membership and counting structures. Membership structures are designed to determine whether or not a specific element is a member of a set. As for counting structures, they are used to measure the frequency of an element or event. Figure 2.6 displays a generic idea of how membership structures work. Typically, an input is processed through hash functions to map it to a specific position within a data structure. This position is then marked as occupied. When a query is performed to check upon the existence of an element within the structure, the same hash functions are applied to the input, mapping it to the same position. If the position is marked as occupied, the element is assumed to be present. Counting structures work similarly, except, instead of just setting the position as occupied they usually store a counter or other additional data

relative to some event. Despite adhering to the same fundamental idea, numerous variations of the described types of structures have been developed, with the aim of reducing collisions, enhancing query performance, improving scalability, optimizing memory usage or simply addressing specific use cases.

One of the said variations of membership structures is the Bloom Filter [2], a probabilistic data structure that uses a bit array and multiple distinct hash functions. Given an element, each of the hash functions maps it to a position in the bit array and sets the bit on that position to 1, meaning that the element is now a member of the bit array. So if an element appears and its hashes all retrieve values of 1, that means that the element was already present in the structure. The main issue with this structure lies in the existence of false positives, as different elements may hash in common positions, thus possibly resulting in bits set to 1 that correspond to the full bundle of positions of an element that may not actually be contained in the set. Increasing the amount of memory allocated for the bit array and using efficient hash functions will help mitigate this issue as more positions are available, hence reducing collisions.

Another membership structure is the Quotient Filter [22], which consists of an array where each position corresponds to a set of bits. It uses a single hash function where the first "n" bits of the output, known as the quotient, indicate the position within the array, and the remaining bits are the data to be stored. Three additional bits are used to indicate the status of the position. The first bit shows whether or not the position is occupied, the second bit indicates if it is a continuation of a previous position, and the last if it has shifted from its intended location. So, when given an element, the quotient indicates its position, the remainder is stored, and if the position is not occupied, it will be marked as such. If another element's quotient is the same as an existing one, it means that it would be placed in the same position. Since the position is occupied, linear probing is performed until an unoccupied position is found, where it can be placed. Once placed, it is marked as a continuation, as it has hashed into the same position as another element, and as shifted because it is not in the position where it was supposed to be. The process of linear probing may be difficult to implement in P4 due to the absence of loops.

The Cuckoo Filter [28] consists of an array of buckets and each arriving element gets hashed by two different hash functions. Each hash function refers to two positions within the array. The algorithm will check if any position is available and if it is, one of them is occupied by the element. If none is available, then the element occupying those positions is replaced, and its hashes are performed to find him another position. This process gets repeated until it either finds a position or a set limit is reached. As is the case with the structure above, this type of behavior typically requires loops or recursion, which may prove a challenge with the constructs available in P4. A way that could not only solve this issue but also mitigate its collisions is the implementation of more than a table to perform the cuckoo hashing and

jumping between tables up to a defined limit. However, collisions would result in multiple read and write operations, negatively impacting the system's performance.

Assaf et al. [37] presents an algorithm named SWAMP, which uses a cyclic buffer known as Cyclic Fingerprint Buffer as a membership structure and a dynamic fingerprint hash table called TinyTable as a counting structure that measures the fingerprints' frequency per cycle. Whenever an element arrives, selected features are hashed in order to produce a fingerprint. In the buffer, there is a pointer that indicates the oldest fingerprint stored. This fingerprint is replaced by the newly arrived one and as this replacement occurs, both frequencies of the fingerprints are retrieved from the TinyTable. The frequency of the removed one is decremented and the frequency of the new one is incremented. Tied to these frequencies there is a counter which estimates the number of distinct elements within the cyclic buffer. Upon the frequency of a fingerprint reaching zero, it means that the corresponding element does not yet exist within that cycle, thus this counter is decremented as there is one less distinct entry. If a fingerprint's frequency reaches one then it means a new element was inserted, which caused the counter to be incremented. Since it is a cyclic buffer, the index pointing to the oldest element is incremented every time a fingerprint is inserted, and, whenever this index value equals the size of the buffer, it means a cycle has been completed, thus it is reset to its initial position. This idea of cycles and the estimated number of distinct elements poses an interesting approach for applying anomaly-based detection techniques during transitions to new cycles.

The presented structures are among the most popular membership structures, but there are multiple variations among them, which typically can improve factors such as membership queries, memory usage, insertion speed and more. Some of them even include counting methods (e.g., measuring the frequency of those entries, rather than just setting a bit to 1), which means that membership and counting structures do not need be mutually exclusive. An example of this is the Count-Min Sketch [8], a counting structure, somewhat similar to the concept of multiple Bloom Filter, except instead of just setting the bit to 1 to assert membership, it modifies a counter value at that position. So, essentially it is a set of arrays, each tied to a different hash function where in each position there is a counter that measures the frequency of that element. When verifying the value of the counters, if they differ, the minimum value should be the one considered. The characteristics are similar to those of a Bloom Filter except it requires more memory usage. However, using multiple arrays further reinforces the reduction of collisions.

2.7 Time series change detection algorithms

Typically, in the light of anomaly-based intrusion detection, it is critical to continuously monitor specific relevant statistics in order to identify potential threats or anomalies. The primary objective is to detect

deviations or fluctuations within these statistics. This is where statistical algorithms become indispensable instruments, as they are able to identify both substantial and subtle changes in the data that could possibly correspond to malicious activity.

Amongst these techniques [14], one of the most popular is *entropy*. Essentially, it serves as a summary of the histogram within a dataset, measuring the dispersion of distribution differences. An increased level of variation in this measurement may suggest the occurrence of anomalies.

Another favored technique is Exponentially Weighted Moving Average (EWMA) [27], which operates as a moving average with assigned weights for each observation. It requires two phases, with the first involving an iteration through all the data to obtain the initial average and establish threshold values. The subsequent iteration is used to determine the said moving average.

Cumulative Sum (CUSUM) [1, 4] relies on the accumulation of observations. It comprises two phases. The initial phase calculates the average of the whole data, while the second involves the addition of the prior value of the CUSUM (initially 0) to the difference between the current value and the average. Historically, every observation has the same weight, which may interfere with its accuracy on more pronounced discrepancies [27].

Adaptative Windowing (ADWIN) [11, 16] operates by maintaining a fixed-size window that holds the most recent data entries. This window is then dynamically divided into two subwindows, and the algorithm continuously calculates the averages of these subwindows. When the average of each window surpasses a certain threshold of difference between them, it triggers an anomaly and discards the older of the two subsets. One of the main issues with ADWIN is that it can accumulate a large amount of stored data if there are no significant changes in the data stream [27]. This happens because it does not trigger the removal of the older subwindow, which can lead to increased memory usage.

Kullback-Leibler Distance [14], also known as *Relative Entropy*, is a mathematical measure that can be used to figure out how one probability distribution has changed with respect to another over time. The idea behind it is to quantify the difference or distance between two probability distributions. As for the *Cosine Distance* [14], the core idea is identical to the Kullback-Leibler Distance except that the calculation resides in the cosine distance between two probability distributions instead.

The Fixed Cumulative Windows Model (FCWM) [16] method employs two fixed-size time windows: a base window that represents normal data and serves as a reference window, and the current window. It compares the data distribution between these two windows using the Kullback-Leibler divergence technique, comparing this difference to a threshold indicative of change drift. Adaptative Cumulative Windows Model (ACWM) [18] operates similarly to FCWM but distinguishes itself through its adaptable window

size, which dynamically expands when the Kullback-Leibler difference between the two windows increases and contracts when it decreases. When contrasted with FCWM, ACWM showcases superior detection speed [18, 27].

The Page-Hinkley Test (PHT) [16] is a sequential analysis method that focuses on detecting variations in the average of a Gaussian signal. It works by maintaining a cumulative sum of the differences between each observation and the average of past observations, adjusted by a factor to limit fluctuations. It also tracks the minimum value of this cumulative sum. If the difference between the current cumulative sum and this minimum value exceeds a predefined threshold, a change is detected. Despite being fast and requiring low memory resources in comparison to other algorithms, every observation holds the same weight, and setting the threshold correctly can be challenging, commonly leading to a high rate of missed detections and false alarms [27].

Overall, according to Chabchoub et al. [27], the EWMA algorithm appears to outperform the others, boasting several key advantages. The most impactful ones include not requiring the storage of large amounts of data and allowing for the assignment of weights to observations. The only drawback is the necessity to iterate twice over the data, which does not allow real-time detection. However, to address this, it is possible to simply set an initial value, and expect the system to adapt over time. In terms of detection, this may cause a few false positives at the beginning but it should quickly stabilize depending on how often the calculation is performed.

Furthermore, it is worth noting that the techniques mentioned above are not mutually exclusive. They can be applied in combination to smooth values, potentially reducing false alarms.

2.8 NIDS developed on P4

When considering the development of solutions in P4, it is important to understand how they can be affected by the data plane restrictions. One key consideration is that network devices have limited memory resources. While an algorithm may respect those limits, it is necessary to guarantee that there is capacity available to deal with future issues or updates. Furthermore, the complexity of the algorithms and data structures must be managed to ensure it does not heavily hinder the system throughput, which could affect the availability of services or applications. This section discusses some NIDS developed using P4 and examines how they operate under these constraints.

With the aim of reducing the load on the system and ensuring scalability, Lewis et al. [47] propose an NIDS known as `P4ID`. It maintains a stateful stage for known ports and a stateless stage for the rest. In the stateless stage, known attack signatures (signature-based) are mapped to P4 tables to determine

whether a packet should be ignored, forwarded normally or forwarded to another IDS. In the stateful stage, a fixed number of new flows are sent to the IDS, while the rest, if they meet a set of conditions, are forwarded normally. According to Tavares and Ferreto [64] and Gao and Wang [54], due to the lack of traffic limitation forwarded to the IDS regarding suspicious flows, overflow situations might take a toll on its performance. Tavares and Ferreto [64] also mention that it may not be possible to apply all known signature rules to tables, since they are stateful structures implemented on the data plane, meaning that they must have size restrictions. Furthermore, the said load reduction is dependent on the predefined set of known ports.

Ndonda and Sadre [38] introduce an approach to intrusion detection with a 2-layer NIDS design. The first layer analyzes and filters packets based on allowlists[4], whereby if there is a match, the traffic is assumed to be legitimate. One of the allowlists belongs to the `Modbus` protocol [96], while the other requires traffic from an initial learning phase. This learning phase considers that a flow is represented by source IP, destination IP, source port, and destination port. It also takes as a basis that the network pertains to an Industrial Control System (ICS) and, therefore, the traffic captured from such a network is representative enough. The flows regarding this traffic are captured and added to the allowlist. After this phase, if a packet does not match an entry of the allowlists, it is sent to the second layer of the algorithm, where it is analyzed by an IDS named `Zeek` [97] and, depending on the result of the analysis, their flow can be added to the allowlist. As in the previous case, there could be instances of excessive suspicious flows, potentially overwhelming the control plane, and, in order to mitigate this issue, the controller was relocated to another host. This algorithm primarily uses allowlists, which are implemented as P4 tables that match flows through lookups.

`P4-ONIDS`, an NIDS proposed by Tavares and Ferreto [64], employs pre-filtering with the aim of reducing the resources used. This is divided into two parts, with the first part involving the selection of rules to be converted into a format compatible with P4 tables, enabling their usage in the data plane. These rules stem from an IPS known as `Snort` [95], and they undergo a process of generalization and duplicate removal. In other words, if there is a rule that can be encompassed by a more general rule, the more general rule takes precedence. After that, they are sorted according to their degree of severity and selected according to a provided size for the P4 table. Flows that match these rules are considered suspicious and sent to an NIDS, in this case, `Snort`. To keep track of these flows, they are temporarily stored in Count-Min Sketches [8] (described in Section 2.6) that estimate the frequency of their occurrence. In terms of resource management, `P4-ONIDS` should perform more optimally than the previously discussed NIDSs

[4]https://www.acm.org/diversity-inclusion/words-matter

in this section, thanks to its use of more generalized rules and Count-Min Sketches.

Ilha et al. [59] presents EUCLID, an anomaly-based mechanism with both intrusion detection and prevention functionalities towards DDoS. EUCLID operates using fixed-size observation windows. As packets arrive, they increment the element count within the current window. Once the count reaches the fixed size, it transitions to the next window. During each window, data about the packets is collected. This data consists of two Count-Min Sketches: one that keeps track of the frequencies of the source IPs and another one for the destination IPs. After every observation window, the Count-Min Sketches have their entropy calculated based on the frequency of each IP. The entropies are then used in an EWMA and in a Exponentially Weighted Moving Mean Deviation (EWMMD) calculations that determine whether or not an anomaly has occurred. To determine if the entropy values are within the expected limits, a prior training phase is conducted, where some thresholds are set. The calculations regarding the decision mechanisms usually involve floating-point numbers, which are not available in P4. To circumvent this, the algorithm performed bitwise shift operations so that the other decimal cases could be considered. Up to this point, the discussed NIDSs mostly rely on rules and signatures which involve their mapping to tables. However, those rules and signatures only served to alleviate other external components. In that aspect, EUCLID adopts a more data plane driven approach, by employing Count-Min Sketches, counters and entropy, EWMA, EWMMD related calculations, which require the data plane to be able to modify them. The code for EUCLID is publicly available[5]. By analyzing it, it is possible to notice a large number of read and write operations which in real scenarios might have an impact on the system's throughput.

2.9 Summary

This chapter introduced the key aspects of data plane programmability using the P4 language, covering the different programmable blocks, the core limitations of the constructs involved and the distinctions between the targets that a P4 program may be deployed to. The chapter also outlined the process of modeling a cyberattack, including the role of port scanning within this model. It discussed the motivation behind port scanning and the core variations of this technique. One particularly challenging variation is the *slow scan*, which, due to its extended duration, requires the system to retain information for long periods so that probes can be correlated, which can be challenging to implement in data planes due to their restrictions in terms of memory. Additionally, the concept of network intrusion detection was explored, highlighting the different mechanisms it employs. A significant focus was placed on anomaly detection, including the algorithms it applies to identify anomalies and the structures that help track elements related

[5]https://github.com/asilha/ddosm-p4

to the statistics these algorithms use. The next chapter presents the proposed solution, which emphasizes anomaly detection with the aim of identifying deviations caused by *slow port scans*, without neglecting other scan strategies.

Chapter 3
Proposed Approach

This chapter provides a detailed presentation and discussion of the proposed solution for network anomaly detection, P4 Cyclic Adaptative Variation in the Occurrence of Distinct Entries (P4-CAVODE). It begins by outlining its structural design and workflow, followed by some considerations that demonstrate how it is optimized to leverage the capabilities of the hardware at use.

3.1 Architecture

An anomaly-based detection system designed to identify port scans must consider all the core characteristics that can define its behavior. That includes the previously discussed header, source and destination multiplicity, and timeframe strategies. Regarding the headers, the strategies explored in Section 2.3.2 consist of combinations of protocols, flags and other parameters. Implementing mechanisms against every single combination that might expose some flaw in the protocol design is unreliable, hence aiming for a generalized solution is more efficient.

As for the targets, every type of port scan always addresses either multiple ports or multiple destination IPs or a combination of both. Thus, having a combination of at least both makes sense as a key parameter to membership structures. Timeframes are related to how long information is kept in the system. A fast scan causes a large number of distinct combinations of destination IP and port to appear in a very short time period. On the other hand, slow scans are a major challenge since its probes are spread over a longer time window, difficulting the correlation between them. As mentioned in Section 2.7, change detection algorithms can be used to identify deviations from normal patterns regarding certain statistics over time. Techniques with weights attributed to past states, such as EWMA, allow for a correlation between the present and the previous states. Thus, each state must contain information regarding the number of keys so they can be used, which leads to membership structures like the ones described in Section 2.6.

Lastly, comes the question of how long should a state last and how often the calculation of the changing mechanism should be performed. The SWAMP algorithm [37], described in Section 2.6, answers both

these questions through its membership and counting structures, a cyclic fingerprint buffer and a hash table that keeps the frequency of each fingerprint.

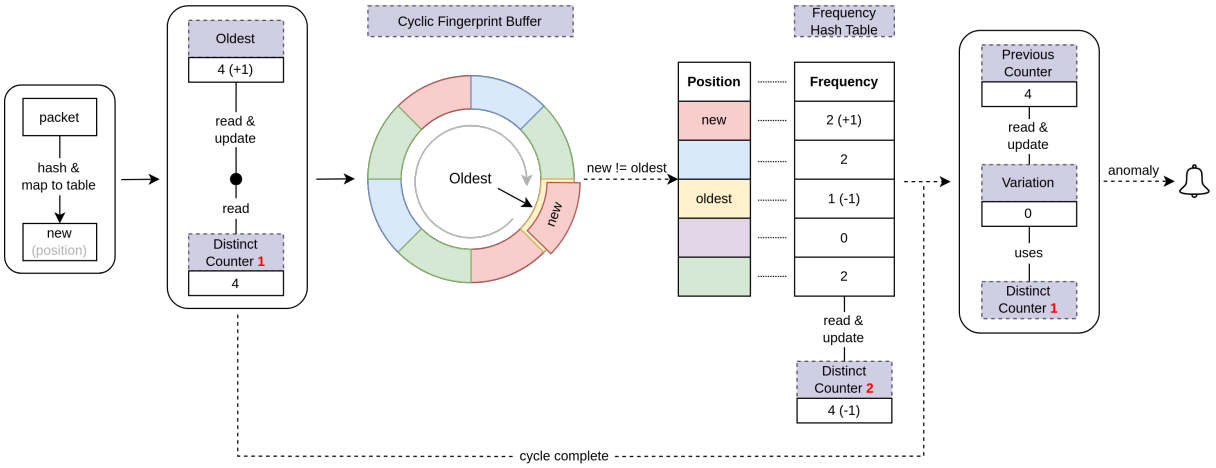


Figure 3.1: P4-CAVODE workflow

As such, the proposed solution takes SWAMP as inspiration with its workflow being represented in Figure 3.1. Essentially, a newly arrived element is hashed and mapped according to the size of the frequency hash table, resulting in a position that acts as a fingerprint. As the membership structure, the cyclic fingerprint buffer stores this fingerprint in place of the oldest stored fingerprint. Since it is a cyclic buffer, data is stored sequentially and in a circular fashion where the index increments with each placement and resets to the beginning once it reaches the buffer's size. This means that during the replacement, the older fingerprints are part of a prior cycle. The frequency hash table is tasked with keeping track of the number of occurrences of each fingerprint per cycle, along with a counter that monitors how many unique fingerprints occurred in that cycle. Whenever the position of the oldest fingerprint reaches the cyclic buffer size, it means that the ongoing state period is concluded and therefore the change detection technique, a weighted moving variation average, should be applied. This variation uses both the distinct entries counters held by the previous cycle and the new cycle. According to a certain condition, explained later in the present section, the variation can be set up to trigger an alarm. This alarm is then sent to the control plane, signaling that an anomaly has taken place in that cycle.

In greater detail, whenever a packet arrives, its destination IP address and port number are combined and hashed to create a position within the frequency hash table limits. This position maps elements to the frequency hash table and takes the role of fingerprint in the cyclic buffer. The cyclic buffer is associated with an index that allows for its rotation. This index points to the oldest fingerprint contained and as this fingerprint gets replaced by a new one, the index will be incremented, thus rotating, until it completes a cycle. Whenever the cycle is completed, the index is reset and the change detection technique is applied,

according to the distinct counter collected during the frequency table process. The size of this buffer can be adjusted to allow for more traffic activity to be held and analyzed at a time.

The frequency table process itself consists of updating the frequency of added and removed fingerprints as they occur according to their positions (fingerprints). This is linked to a counter that keeps track of how many fingerprints resulting from the different combinations of destination IP and destination port are present within each cycle of the cyclic buffer. A frequency with a value of zero means that an element has not occurred so far within the cycle. So whenever a frequency goes from zero to one, it means that a new fingerprint is present in the current cycle, thus it causes the distinct fingerprint counter to be incremented. The opposite event, a frequency going from one to zero, means that the element was part of the previous cycle and was replaced by a new one, hence the distinct fingerprint counter gets decremented. While an element may appear more than once per cycle, resulting in a higher frequency value, this does not affect the counter since it only concerns unique fingerprints. Additionally, if the new and the old fingerprints are the same then the entire frequency table process and the update of the distinct counter are skipped. This is because updating the frequency would involve a decrement followed by an increment, resulting in no actual change to the stored value and thus not affecting the distinct fingerprint counter. Since the frequency table has a limit, two different combinations of destination IP and port may map to the same position causing a collision, which means that the distinct counter is but an estimate of the count of these combinations. Adjusting the table size may mitigate collisions by expanding the number of available positions for the mapping of different keys.

Aiming to determine if any anomaly might have occurred or not, a change detection technique is applied at the end of each cycle. The process used is a weighted moving variation average and works as follows:

$$\begin{cases} obs(t) = |dist(t) - dist(t-1)| \\ var(t) = (1-\alpha) \times var(t-1) + \alpha \times obs(t) \\ (dist(t) - dist(t-1) > \gamma \times var(t-1)) \implies \text{ALARM}, \qquad dist(t) > dist(t-1) \end{cases} \tag{3.1}$$

The variation equation is based on EWMA (see Section 2.7) and it serves to highlight and adapt to the changes in the distinct entries between cycles. As such, the observation $obs(t)$ is the absolute difference between the current count of distinct entries, $dist(t)$, and the one from the previous cycle, ($dist(t-1)$). The current variation $var(t)$ is calculated based on the said observation and the previous variation, $var(t-1)$. Each time a new variation is being calculated, the cumulative past observations are multiplied by a weight α, meaning that every past observation is being reduced over and over again,

hence the exponential factor in EWMA. This allows the solution to adapt to changes over multiple cycles while taking past variations under consideration, reducing their importance as they get older.

To identify an anomaly in those changes, a message is sent to the control plane. This message is called an alarm and it is triggered whenever the difference between the current distinct count and the previous distinct count exceeds the previous variation. However, as mentioned earlier, a port scan involves multiple ports, addresses, or a combination of both. This leads to an increase in the number of distinct combinations of destination IP and port that constitute the key at use. As such, a decrease in the distinct counter of a cycle compared to the one of the previous cycle should not trigger an alarm. Therefore, the defined alarm condition requires that the current distinct counter must be greater than its predecessor.

Since it is a moving average technique and adapts itself over time, the initial values for both the variation and observation considered are zero. This means the system will require a few cycles to stabilize, causing the alarm system to identify this initial step as anomalies. The variation's γ and α (weight) are values fixed between zero and one, and their adjustment to certain environments may yield better results. The γ is relative to the alarm condition and defines how much higher the current variation has to be in comparison to the previous one. The lower the value, the more prone the alarms are to be triggered since it would only require fractions of the previous variations to activate them. The α pertains to the weight given to the most recent observation, dictating its significance compared to previous ones. A lower α assigns more weight to past observations, lessening the fluctuations caused by recent events. Conversely, a higher α gives more weight to recent observations, making the equation more sensitive to changes in the data. Given the behavior of port scan, namely the increase in the observation during its occurrence, both γ and α should tend to the higher side.

3.2 Considerations

3.2.1 Fingerprint and data structures

The described way of using the position as a fingerprint allows the preservation of memory since the actual fingerprint is not being stored. While one could argue that storing the actual fingerprint could help reduce the number of collisions, as per some of the structures priorly described in Section 2.5.5 and Section 2.6, the trade-off in terms of memory used and operations to deal with those collisions would hinder the performance. Thus, an estimate number is used, since the content of every stateful structure is always refreshed, which also eliminates the need to perform operations to delete or even reset them. A frequency will be reduced to zero if its associated element does not occur in a cycle, as it will be decremented as

the cycle passes, meaning that the estimated number of distinct entries is also updated and therefore independent from past iterations.

The TinyTable used by SWAMP [37] is a structure that dynamically adjusts the memory allocated to each bucket according to its load. However, P4 does not support dynamic structures, so, even though fewer positions may be being used at a given moment, the total memory would still be allocated, so in the end the structure used ended up being a hash table with the frequency of each fingerprint, where the fingerprint is simply the position where it is placed.

Port scans focus on probing multiple ports. However, just using the destination port as a key would not allow the system to detect scans that consider repeated ports across multiple targets because they would be mapped to the same position of the table. By using a combination of destination IP and destination port as key, it is possible to tackle any target-dependent port scan strategies like horizontal, vertical, strobe and block scans (see Section 2.3.2). This key also allows for the detection of anomalies caused by distributed scans since it focuses on its destination rather than the source. Additionally, the strategy behind P4-CAVODE does not rely on the analysis of flags which makes it efficient against header-based port scanning techniques.

On the topic of timeframe, both fast and slow port scans are expected to cause a variation in the number of distinct entries since they probe multiple ports. The difference lies in the fact that fast port scans have a very low delay between probes, meaning that the expected number of distinct entries should significantly increase over the course of very few consecutive cycles. In counterpart, in slow scans, the distinct counter is expected to vary much less as only a low number of probes may be present within each cycle, according to the delay used. Since probes are more distributed over time, the number of generated alarms should be much larger in comparison with fast port scan.

3.2.2 Regularity of applying change detection techniques

Anomaly-based detection approaches often face a dilemma regarding how often should the anomaly detection technique be applied. Applying it too early may not consider enough data for a deviation to be detected, and doing it too late may not be ideal in real-time approaches, as it would cause delays in detections and could also be costly in terms of memory, depending on the structure used. Most common solutions choose to either use frequency windows or time windows.

On one hand, time windows could ease out the detection capabilities for their respective period, as they may be able to capture more activity within it. On the other hand, adjusting their parameters may be quite a challenging task for security managers, as the traffic may vary significantly. This would require

the data structures to be able to endure the traffic load, which should be done dynamically to ensure the memory used does not surpass the system limitations. One could explore solutions that dynamically adjust it, but another issue arises: the memory in P4 cannot be dynamically allocated. While a maximum amount of memory could be allocated for the time window and a smaller or equal value defined to represent the current usage, it would require an overall larger allocation to deal with higher traffic load periods, potentially reducing the resources available for further features in the system. Outside those periods, the reserved memory would remain unused, which is not memory efficient.

Moreover, considering the proposed solution, if a time window were to be used and, at some point, the traffic became more concentrated, this could result in an increased amount of collisions that could reduce the quality of the estimation value held by the distinct counter. Additionally, an increase in the time before applying the variation calculation translates to an increase in the delay between what would be an anomaly and the generated alarm, while reducing it could lead to a possible decrease in the accuracy by missing legitimate services, that later cause an alarm. Defining this maximum amount of memory would have to consider both this detection delay and the maximum number of packets that it could withstand at a time.

On those aspects, frequency windows are more direct to adjust as they focus on counting events (e.g., number of packets), moving on to a following window whenever the said count reaches a defined threshold. Even if in a period there was an increase in traffic activity, the data held by structures would be limited to this threshold, allowing a more fixed control over both memory usage and the actual activity analyzed per cycle. In this case, the window used is a frequency window that comes in the form of the cyclic buffer, where a cycle is indicative of when the anomaly detection technique should be applied. Adjusting its size according to the typical traffic scenario of a desired system is a must, since if it is too small, every other window may include different services and cause alarms unrelated to the object of detection, and if it is too big, it may ruin the intended purpose, which is close to real-time detection.

3.2.3 Implementation

P4-CAVODE features two implementations: a software version, where the target device is emulated by BMv2, allowing the P4 program to function as NFV, and a hardware version, with Netronome NFP-4000 Agilio CX 2x10GbE SmartNIC as the target device.

To successfully implement P4-CAVODE, six data structures are deemed as essential. These structures ended up being under the form of the extern stateful structures known as registers, as allocating memory for each one is necessary. The most apparent structures include the cyclic buffer and the frequency table,

which address membership and counting, respectively, along with the counter that estimates the total number of distinct entries and the change detection technique addressed as moving variation. The other two are a value that indicates the index of the oldest element — the next element to be replaced within the cyclic buffer - and another counter that stores the estimated number of distinct entries of the previous cycle, required by the moving variation equation. This previous distinct counter is updated every time the index of the oldest element gets reset, due to the beginning of a new cycle.

Regarding the calculation of the variation value, one relevant challenge arose: the lack of floating-point arithmetic support in P4, which did not allow the direct multiplication by the weight α. Tackling this limitation involved approximating the intended α, through bitwise shift operations. In the performed tests, the chosen α was 0.7. That would be roughly:

$$value \times 0.7 \approx value \times \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256} + \frac{1}{2048}\right) \tag{3.2}$$

$$= value \times \left(\frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^7} + \frac{1}{2^8} + \frac{1}{2^{11}}\right) \tag{3.3}$$

$$= (value >> 1) + (value >> 3) + (value >> 4)$$
$$+ (value >> 7) + (value >> 8) + (value >> 11) \tag{3.4}$$

However, when performing these calculations there would be precision loss amidst each operation. Fingerhut [79] suggested performing a left shift of, at least, the largest shift factor over the value and after the operations finished, perform an opposite right shift, thus diminishing the issue at hand. Through this logic, the precision of the equation could be improved by increasing the shift factor.

Simplifying the variation equation, to reduce the amount of bitwise operations:

$$var(t) = (1 - \alpha) \times var(t-1) + \alpha \times obs(t) \tag{3.5}$$

$$= var(t-1) - \alpha \times var(t-1) + \alpha \times obs(t) \quad \text{(expanding } (1-\alpha)\text{)} \tag{3.6}$$

$$= var(t-1) + \alpha \times (obs(t) - var(t-1)) \quad \text{(applying distributive property)} \tag{3.7}$$

Thus, the actual process ended up being more akin to:

$$\begin{cases} obs(t) = |dist(t) - dist(t-1)| \\ value = (obs(t) - var(t-1)) << 11 \\ intermediate = (value >> 1) + (value >> 3) + (value >> 4) \\ \qquad\qquad + (value >> 7) + (value >> 8) + (value >> 11) \\ var(t) = var(t-1) + (intermediate >> 11) \end{cases} \tag{3.8}$$

Onto the memory required by each of the said structures, the size of both the cyclic and frequency table may be adjusted in many ways. As discussed in Section 3.2, the number of cyclic buffer entries should be dependent on the throughput under consideration. Since the fingerprint is a position of the frequency table, each entry of the cycle buffer will have to allocate memory enough to hold that fingerprint, meaning that it is directly tied to the size of the table. Conversely, each position of the frequency table holds its respective frequency, which can only be at most the maximum index of the cyclic buffer. As for the table number of entries, in the worst case, every fingerprint in the cyclic buffer will be different, meaning that at most it would need as many positions as the cyclic buffer. Realistically, there is no need for that for two reasons. First, the entity deploying the program should be aware of the number of services within its system, meaning that a fixed maximum amount of services should be expected. Second, a port scan covers multiple different keys. This means that while some of these keys may generate the same fingerprint and cause collisions on the fingerprints of the said services, other probes will still affect the distinct counter, causing a deviation and triggering an alarm. However, despite not requiring the same size, reducing the frequency table too much will increase the risk of collisions, since it also increases the chance of different keys associated with port scanning probes to be mapped to the same position in the frequency table. This impacts the distinct counter, hindering the detection capacities of the solution. As for the counters and the oldest element structures, they will require at most enough memory to make up the maximum index of the cyclic buffer, while the moving variation size will heavily rely on the number of decimal places desired to preserve the precision of the algorithm.

Assuming X as the intended entries for the cyclic buffer and Y for the frequency table, $\log_2(X) = N$ bits and $\log_2(Y) = M$ bits will be required per entry, respectively. Therefore, the resulting cost per structure is as shown in Table 3.1:

Table 3.1: Memory cost per structure

Structure	Cost in bits
Cyclic buffer	$N \times M$
Frequency table	$M \times N$
Oldest index	N
Distinct counter	N
Old distinct counter	N
Moving variation	$N + precision\ bits$

Regarding the portability factor, up until this point, since both implementations feature the same core architectural model, the constructs and the overall logic behind the implementation are essentially the same. However, two aspects named in Section 2.2.1, concerning the hardware, proved to be a major challenge. The first and most challenging was the registers not being synchronized with the in-hardware flow cache. According to the documentation provided with the hardware, disabling the cache-flow option of specified actions through the use of a directive would solve this issue. Despite that, when adding it, while the program compiled successfully, no changes were perceivable in the behavior of the final program. In Section 2.1, it is mentioned that in the compilation workflow of a P4 program, the back-end compiler uses the IR to produce a binary that is loaded onto the target data plane. Regarding the Netronome Agilio SmartNIC, its back-end compiler first converts the IR into a C-code that is later compiled by the Netronome Flow C Compiler (NFCC), so the binary file can get produced [53]. Yet, when comparing both versions of the program with and without the said directive, their resulting C-code held no differences, meaning that the compiler was ignoring those directives. As such, the way to overcome this compilation issue was by manually modifying the desired action in the C-code to include the flag that disables this cache option.

The other relevant challenge was caused due to the difference in the threading capabilities of both targets. BMv2 uses a single thread for the ingress pipeline, not requiring the application of any mechanism for synchronization since the packets are processed sequentially. In contrast, the Netronome SmartNIC relies on multi-threading, and for that reason, without proper critical section mechanisms, the reading and writing operations performed by different threads would lead to data corruption and race conditions, which would inherently ruin the P4-CAVODE workflow. To address this issue and take actual benefit from the characteristics of such hardware, defining extern locks in C became fundamental. However, the employment of general locking mechanisms proved to significantly degrade the solution performance, which prompted the adoption of a more granular locking strategy. The implemented locks were based on the FastReact repository [77].

For these locks to provide any discernible improvement, the critical section of each structure, where their memory is accessed for both reading and writing, should be isolated as much as possible from any operation regarding another structure.

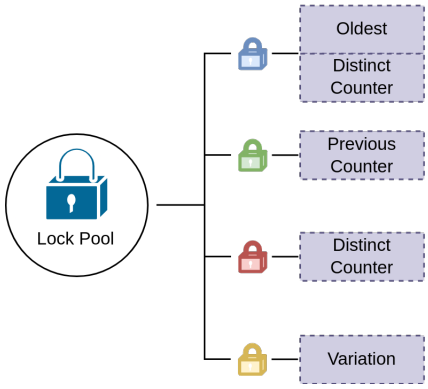


Figure 3.2: Lock granularity in single-element structures

Thus, each data structure within the system was evaluated for its locking requirements based on its access patterns and composition. For single-element structures, such as the oldest index, the two distinct counters, and the moving variation, a dedicated spin-lock was allocated to each. This approach ensures that operations on these individual elements do not unnecessarily block access to unrelated components. An exception to the said isolation of operations is the distinct counter, which is not only read before being updated but also at the index critical section to ensure that the value used in the variation calculation is the one read only up until that point, hence not being affected by any changes caused by incoming packets, as depicted in Figure 3.2. Technically, one would only require it to be read at each reset, but due to limitations in P4, this method ended up being more efficient, in terms of throughput.

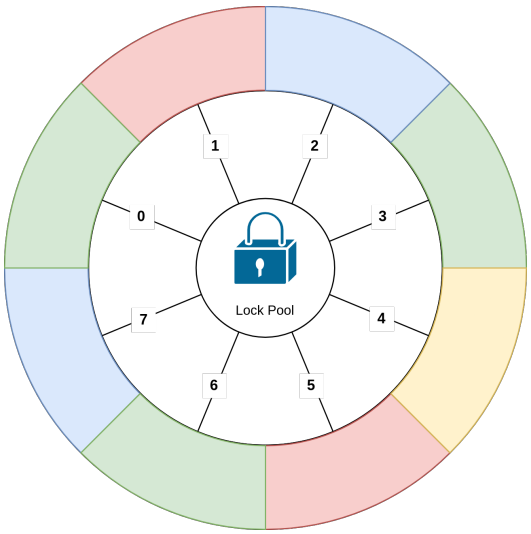


Figure 3.3: Lock granularity in the cyclic cuffer

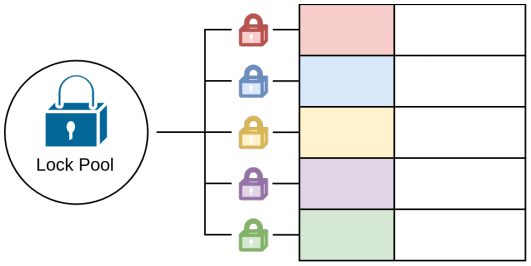


Figure 3.4: Lock granularity in the frequency table

For the more complex structures such as the cyclic buffer and the frequency table, their partitioned nature eased out this process, allowing a finer-grained locking scheme to be adopted. Each entry within these structures was associated with its own spin-lock, as seen in Figures 3.3 and 3.4. By doing so, operations on different entries could proceed in parallel without imposing a lock on the entire structure, improving the overall performance.

Whenever read operations are performed, they are stored in temporary variables or metadata, exclusively associated with each packet. This way, the operations following a specific structure access can use the stored value, meaning that they do not depend on the continued exclusivity of access to the memory location, hence the system can release the locks faster. Therefore, not only data integrity is safeguarded during concurrent modifications but there is also an optimization of the system's throughput by enabling more granular concurrency control.

Lastly, the alarm generated consists of a function called *digest*, which sends a message to the control plane. This message contains information regarding its timestamp, the variation and its required parameters, namely the distinct count and its difference compared to the previous distinct count. Here, the timestamp between the software and hardware versions also had to be modified, given the differences in the used formats mentioned in Section 2.2.1.

3.3 Summary

This chapter introduced the proposed solution, P4-CAVODE, a lightweight anomaly detection system focused on identifying slow port scans. It detailed the overall design, the functions of each component, and the considerations behind its development. Although the primary target of detection is slow scans, P4-CAVODE also considers other strategies outlined in Section 2.3.2. For instance, fast scans are detectable as they tend to generate bursts of distinct entries in each cycle, thereby triggering alarms. Moreover, P4-CAVODE does not rely on the analysis of header strategies, making it independent from flags or other fields. By using the target IP address and port as keys, it also accounts for variations concerning the target multiplicity. Furthermore, the chapter addresses the implementation details of the algorithm, highlighting the differences between software and hardware implementations. Additionally, it discusses the critical section mechanisms designed to leverage the multi-threading capabilities of the hardware. In the following chapter, the P4-CAVODE's performance and detection capabilities will be demonstrated, showcasing how effectively it utilizes hardware capabilities.

Chapter 4
Evaluation and Results

This chapter addresses the application of the P4-CAVODE across different traffic scenarios, aiming to highlight key factors expected from such a detection system and how it meets the objectives listed in Section 1.2.

4.1 Test environment and overview

To evaluate network event detection solutions, two primary methods are typically used: generating custom traffic or utilizing existing datasets. Creating a dataset can be a challenging task, mainly because its traffic must resemble real-world scenarios relevant to the developed system. Achieving this usually requires an understanding of common behavior patterns and the usage of certain guidelines, which can result in a significant investment of time. Consequently, utilizing existing datasets is often more practical, as they may meet those requirements or actually correspond to real activity. Ring et al. [48] elaborated a survey in 2019, containing information regarding multiple datasets that can be used in the field of intrusion detection, which conveniently mentions the inclusion of *reconnaissance* techniques, namely port scan. The initial tests considered the usage of the said datasets that included port scan. However, most of them were not ideal due to a combination of some of the following factors:

- Unavailable for download;
- Lack of proper documentation;
- Very few number of probes related to port scan;
- Lack of slow port scan;
- Aggregate traffic;
- Bidirectional traffic and location of the capture point - P4-CAVODE is intended for deployment on a device located at the network's edge. As such, it focuses on capturing and analyzing traffic

relevant to external interactions. Consequently, the communication between internal services and their responses to external queries are not within the scope of analysis, since these services are deemed trustworthy.

Out of the tested datasets, WISENT-CIDDS-002 [36] featured the most desirable characteristics for the proposed implementation, namely because its abnormal activity solely consists of port scans, encompassing a diverse range of probing strategies, including `UDP`, `FIN`, `PING`, `ACK` and `SYN` scans as well as some of their corresponding versions as slow scans, the object of interest. This dataset comprises 2 weeks of unidirectional flow-based traffic, simulated in a small business environment, covering interactions with four services: *Mail Server*, *Backup Server*, *Internal Web Server* and *File Server*. The entities associated with those services execute scripts that follow certain guidelines in order to ensure similarity with real traces. An examination of the topology used in WISENT-CIDDS-002, along with an analysis of its interactions with various entities, revelead a need to filter the traffic. This necessity arose because the dataset includes traffic from internal nodes, whereas as mentioned earlier, P4-CAVODE only concerns traffic captured on the network's edge. After filtering the data to focus exclusively on external traffic, while retaining information about the attacker, an internal entity, the resulting sample size ended up being insufficient for a proper evaluation of the proposed solution. Other datasets are also affected by a part of the aforementioned factors, and, while these can be addressed by applying filters, the resulting traffic is too small to be used. This led to a search for a more controlled environment via generation allowing for a more deep analysis of the solution capabilities towards multiple scenarios. To achieve this, the topology of the said dataset was used as the basis for the test environment, and a test setup was crafted to emulate the mentioned active services within a cloud infrastructure, utilizing the Netronome NFP SmartNIC.

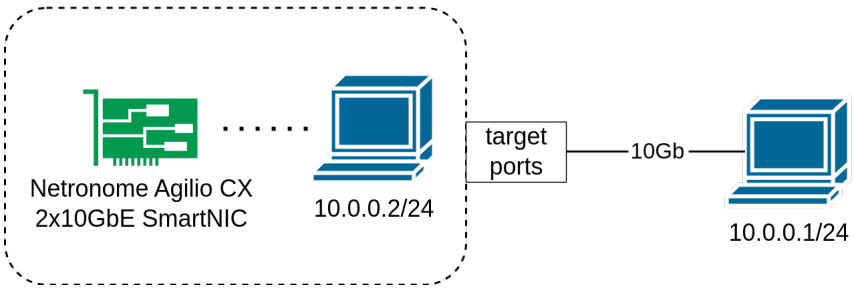


Figure 4.1: Test scenario

The proposed detection system considers as keys the combination of destination IP and destination port. Assuming the representation of each server by a host, this would mean that there would be four distinct destination IP, thus four distinct keys. However, since the destination port is also a factor, having a single destination IP and four destination ports still results in four distinct keys. This means that a

single machine can be used to simulate the four services, as the fingerprints associated with those keys will still yield the same number of keys regardless of whether or not the IP was the same. As for the normal traffic and the attack traffic, both can be performed by another host as no information regarding the source is kept. Both the decisions that lead to this design significantly streamline testing procedures, as all active services can be hosted to different ports on a single machine, with traffic generated from another, allowing for a better deployment and evaluation processes, which results in the test scenario described in Figure 4.1.

The tests performed in an early stage focused on the software implementation utilizing *Mininet* for emulation. *Mininet* [86] is an open-source tool that allows the design and emulation of network environments on a machine. With this tool, one can customize topologies and the behavior of its typical entities, such as switches, routers and hosts, thus presenting itself as flexible and cost-effective for developing and experimenting with network applications and protocols without the need for physical hardware [60]. Given these features, it is commonly used in the development of P4-related programs, with software-based targets such as BMv2. In this case, *Mininet* was used to emulate the hosts and the switch where the P4 implementation was deployed. This served as a validation of the proposed solution's concept before moving to the final tests, which concerned the hardware implementation where the deployment was on a host equipped with the Netronome Agilio CX 2x10GbE SmartNIC.

For these tests, two tools were required for the traffic generation: one for normal activity and another for port scan. Regarding the first one, the normal activity, *D-ITG* [76] was initially considered. It is basically a tool capable of generating customizable network traffic for various protocols. It allows precise control over traffic characteristics such as packet size, inter-packet intervals, and traffic distributions (e.g., uniform, Poisson, exponential, etc), among other features. This makes D-ITG suitable for testing and evaluating network performance under different conditions. With it, in order to introduce variability in the active services' packets per second, and consequently activity per cycle, a Poisson distribution was used. However, when measuring the throughput and packets per second using *iPerf3* [85], a network testing tool for bandwidth and performance analysis, a greater diversity in the number of packets per cycle was observed when compared to *D-ITG*. Therefore, it was decided to use *iPerf3* for the testing instead. As for the port scan, *Nmap* [89] was the utilized tool. It consists of a network scanner, that, in the context of port scanning, offers a considerable array of options regarding the different strategies explored in Section 2.3.2. In short, its purpose is to help test the resistance of NIDS to the various port scanning techniques presented.

Onto the test scenarios, since P4-CAVODE does not perform any analysis on the header fields, it

eliminates the need to test the multiple strategies mentioned in Table 2.1. However, considering the timeframe, both slow and fast port scans represent distinct scenarios. The slow scan, the main and default scenario, is carried using the *Nmap* command, which performs a TCP `SYN` scan that sends probes to the 1000 most commonly used ports across the internet. However, to actually portray the slow part, it is necessary to add a delay between the interactions with each port. This can be done using the `--scan-delay <value>`, in which the `value` corresponds to the delay intended. As such, the tests consider 10, 20 and 30 seconds as the delays between probes to create a slow scan. On the other hand, the fast port scan scenario does not require these time delays between each probe. The probes to each port are done sequentially, meaning that after it finishes interacting with a port, it starts scanning another one right away.

Regarding source multiplicity, the used key does not depend on the source, which is another reason why the topology is designed around a single source host. As for target multiplicity, the key format ensures that requests to multiple targets can be handled by the system. However, an interesting situation arises when there is a service that may receive requests way less often than others. In real scenarios, not every service will be constantly accessed. For example, a service providing analytics reports is unlikely to receive requests with the same regularity as a browsing service. From P4-CAVODE perspective, this means that such service might not be present in every cycle, potentially triggering false positive alarms whenever it does appear. As such, this poses one of the scenarios, where one of the services receives requests sporadically, specifically every five minutes. These requests come from a default *iPerf3* command, which involves sending traffic over a period of ten seconds. The remaining scenario aspects involve tests with slow scan and fast port scan activity and their respective parameters.

Despite the links in the topology supporting up to 10 Gbps, tests unrelated to throughput measurements are conducted at a rate of 1 Gbps for a more controlled selection of the structures' tested parameters, hence leaving 0.25 Gbps per service. The analysis of the throughput is performed on both BMv2 and the Netronome SmartNIC and it includes two results, one concerning a single service and another one concerning the four services in the topology. This test seeks to gauge how much throughput can be achieved with the solution deployed and to evaluate how well the system scales when the number of available active services is increased. All scenarios begin with two hours of normal activity, followed by their respective scan activity, and then conclude with another small period of normal traffic. This results in a total duration of 8000 seconds for the fast port scan, and for the slow scan, depending on the delays used of 10, 20, or 30 seconds, the duration can be 18000, 28000, or 38000 seconds, respectively. The sporadic service scenario operates under the same conditions, depending on the port scan activity

chosen. Each test is individually generated with its respective structural and port scan parameters and not just a replay of the same traffic capture with just an adjustment in those parameters.

When P4-CAVODE is deployed, a first alarm is always bound to happen since initially, all the structures are empty or assigned a value of zero, including the variation. This happens because the first cycle will go from zero distinct elements to a positive value, causing the variation to trigger an alarm. Considering the active services present in the topology, ideally, a cycle should cover their four fingerprints so that fewer variations could cause alarms to occur. Regarding the executed port scan, out of the 1000 default ports scanned, the ports associated with the active services were excluded from the tests to improve the clarity of the analysis. This should leave 996 expected probes. However, in the host, the port 22 was left open, which caused the prober to send an additional `ACK` and `RST/ACK`, leading to a total of 998 packets.

In short, the employed test scenarios and their respective structure parameters are represented by the Tables 4.1 and 4.2, respectively.

Table 4.1: Test scenarios

Scenario	Delays
Slow scan	[10, 20, 30]
Fast scan	No delay used
Sporadic service	[No delay used, 10, 20, 30]

Table 4.2: Test scenario's parameters

Parameter	Values
Cyclic buffer size	[400, 450, 500, 550, 10000]
Frequency table size	1000
Variation's α	0.7
Variation's γ	1
Variation's precision bits	11

4.2 Results

This Section presents the results derived from the test scenarios in Section 4.1. It focuses on evaluating the performance of P4-CAVODE across several key aspects, essential for assessing the effectiveness of an anomaly detection system, particularly when implemented on data plane level. These aspects include: detection latency, which refers to how long it takes for alarms to trigger whenever any activity that would cause an anomaly occurs; accuracy, in the sense of the number of alarms caused by port scans compared to the total number of alarms; achieved throughput; scalability on multiple services.

4.2.1 Detection latency

Each incoming packet results in a filled entry in the cyclic buffer. The cyclic buffer size indicates how many packets each cycle contains before it checks if an anomaly has occurred. For instance, Figure 4.2 displays the featured number of packets per second at 1 Gbps, over two minutes in a scenario without scan activity.

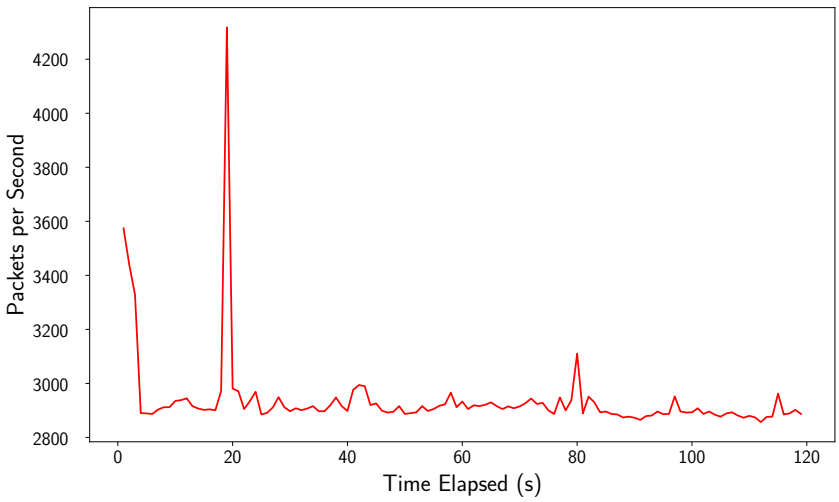


Figure 4.2: Packets per second

Considering the cyclic buffer sizes used in the tests, it is possible to deduce that except for the largest size, 10000, multiple cycles occur every second, given that the value of packets per second is way higher than the sizes used at tests. Anomalies are only detected at the end of each cycle meaning that any activity that causes an anomaly has a detection latency based on the time it took to complete the cycle. The average of those packets per second is around 2936 packets and, consequently, the approximate time to complete a cycle is represented in Table 4.3.

Table 4.3: Approximate time to complete a full cycle

Cycle length	Approximate time to complete a cycle (s)
400	0.136
450	0.153
500	0.170
550	0.187
10000	3.406

These values are not necessarily the same as the detection latency but the average maximum detection latency, given that the packets that cause the anomaly do not necessarily take place at the beginning of the cycle. Regardless of the cyclic buffer size, the approximate times to complete a cycle are shorter than the delays used between scan probes in the slow port scan scenarios, therefore, at most, only one probe can occur per cycle. Similarly, in fast scans, if the delay between probes is higher than the cycle duration they will exhibit the same behavior. However, as in the test parameters, these scans usually do not use any delay, making most values fall in the same cycle or consecutive cycles.

The smaller the cyclic buffer size used, the lower the detection latency of an anomaly. Conversely,

an increase in the size corresponds to an increase in the odds for every active service in the system to be contained, meaning that there is a trade-off between detection latency, memory used and false alarms caused by missing out on such services. This is why it is important to adjust the values, so that the detection may be as close as possible to real-time, allowing countermeasures to be applied and prevent a possible subsequent attack. Despite the approximate time of the cyclic buffer size 10000 being much larger than any other sizes used, it still is not high enough to hinder such prevention measures. On the subject of subsequent threats, while their sources are not the focus of P4-CAVODE detection, a prevention mechanism could be implemented to monitor source IPs, allowing administrators or Controllers to take preemptive actions.

4.2.2 Accuracy

Slow port scan scenario

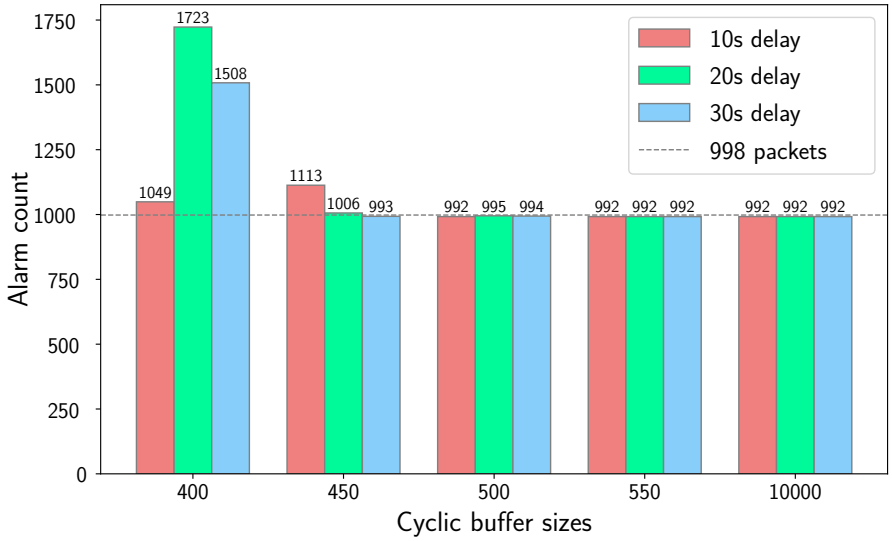


Figure 4.3: Number of alarms per cyclic buffer size

Figure 4.3 depicts the number of alarms for each probe interval used across the different cyclic buffer sizes. As addressed in Section 4.1, 998 packets related to port scan are expected, to which the reference line points. Overall, the lowest cycle lengths, namely 400 and 450, feature a higher number of alarms when compared to the probes sent.

On the other hand, the sizes 500, 550 and 10000 are not only much closer but also below the reference line, with the latter two sizes having no differences at all between them. In some sizes, even though the period of the test gets larger based on the probe delay used, smaller delay values register a higher number of alarms than some larger delays. An example is the cyclic buffer of size 400 generating

more alarms with a 20-second delay than with a 30-second delay, despite the latter taking place over a larger timeframe. The reason for this is linked to each test being generated individually and not just a rerun of a default traffic capture. Repeating the tests leads to abrupt changes in the results, with the order of magnitude of the difference in the number of alarms increasing the smaller the size goes. However, doing so for the three largest sizes barely results in any change.

As noted on Section 4.2.1, each active service receives a very large number of packets in a short time span and, depending on the order of the received packets, a fingerprint associated with a service can sometimes not be part of a cycle. When those services reappear in an upcoming cycle, an alarm is triggered, but since these alarms are linked to active services, they are false positives. This is why smaller cyclic buffer sizes generate more alarms above the reference line, as they often miss including all active services, leading to false positives. In the end, increasing the cyclic buffer size approximates the counts up to the point where both the highest cycle lengths display the closest values in comparison to the reference. Nonetheless, this is not enough to determine whether their generated alarms are false positives or true positives, or if they indeed exhibit higher accuracy. To do so is necessary to analyze the values of the distinct counters associated with the said alarms and if the time passed between those alarms is similar to the probe delays used.

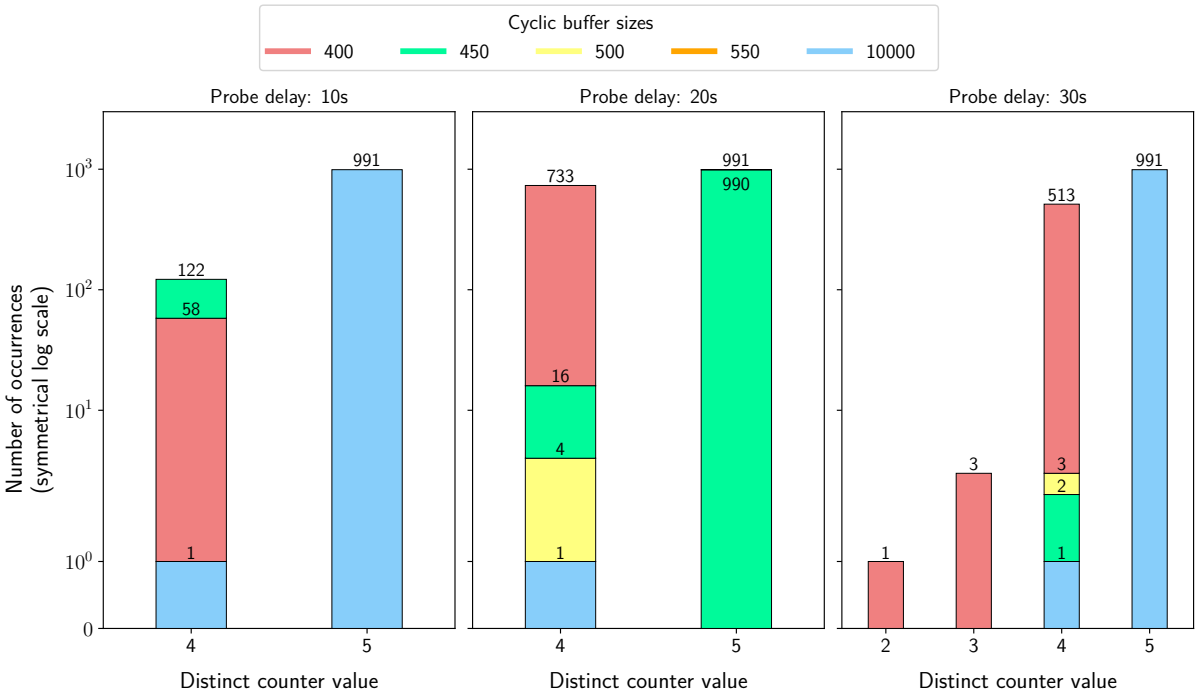


Figure 4.4: Distribution of distinct counter values for different cyclic buffer sizes and delays

Table 4.4: Distribution of distinct counter values for different cyclic buffer sizes and delays

Cyclic buffer size	Distinct counter value											
	2	3	4	5	2	3	4	5	2	3	4	5
400	0	0	58	991	0	0	733	990	1	3	513	991
450	0	0	122	991	0	0	16	990	0	0	2	991
500	0	0	1	991	0	0	4	991	0	0	3	991
550	0	0	1	991	0	0	1	991	0	0	1	991
10000	0	0	1	991	0	0	1	991	0	0	1	991
Delays:	10s				20s				30s			

Figure 4.4 and Table 4.4 show the number of occurrences of the distinct counter values associated with each alarm according to the different cyclic buffer sizes and port scan probe delays used. As addressed in Section 4.1, ideally four services would be within each cycle, hence four distinct fingerprints, and each port scan probe causes an additional fingerprint, which makes the variation trigger an alarm. This would mean that besides an initial alarm with four as the distinct fingerprint count, the other alarms should be triggered when a port scan probe occurs, causing the distinct count to reach five. Since the probes are spaced and multiple cycles pass between them, the variation normalizes, so that when the following probe happens it will trigger an alarm again.

Out of the data represented, only the cycle lengths 550 and 10000 have a single occurrence of four and all the remaining of five. The remaining cyclic buffer sizes seem to have several occurrences of four meaning that some of their cycles do not include all the services, so when it does, it can occasionally trigger false positive alarms. When the used delay is 30 seconds, the distinct counter values associated with the smallest cyclic buffer size, 400, range from two to five, meaning that at some point there was even a window where only one service was present and three windows where only two services were present. Overall, It is possible to notice that the higher the time delay and the smaller the cycle lengths, the more alarms that deviate from the ideal scenario are generated since it is tied to a larger time span and smaller odds of including every service.

Given that 996 ports were probed and considering alarms with counters of value five, each cyclic buffer size can effectively generate true positive alarms in a nearly one-to-one ratio with the number of probes. However, observing the number of counters of value four, it is clear that smaller buffer sizes tend to produce a higher number of false positives. That said, achieving such a high true positive ratio is not strictly necessary, as it reflects the port scan only after its completion. Instead, a Controller should analyze alarm messages in real-time to detect and identify the scan as it takes place. For example, this Controller

could monitor the number of alarms associated with specific sets of hosts within defined time windows and identify deviations from normal patterns by employing time series change detection algorithms, discussed in Section 2.7.

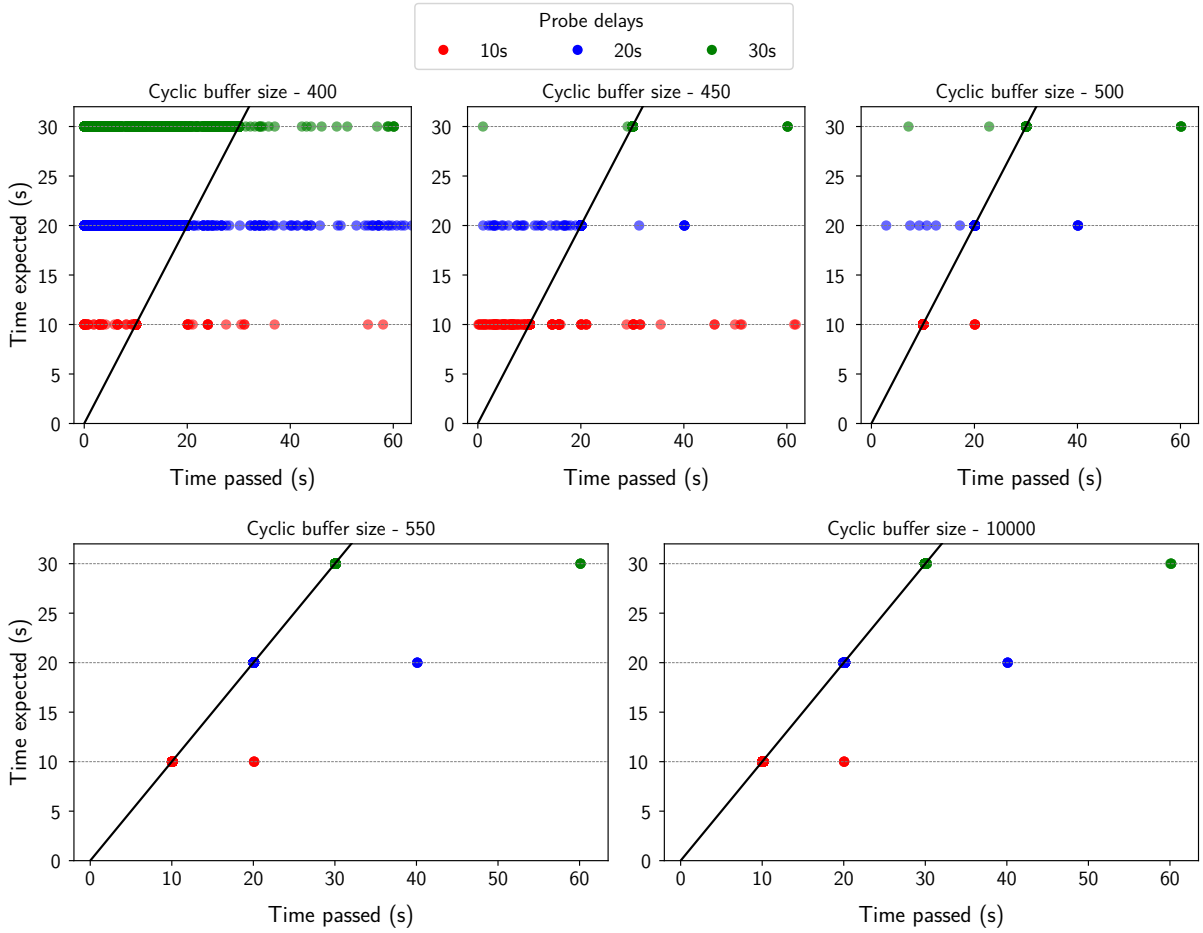


Figure 4.5: Comparison of the expected and the actual time passed between each alarm

Regarding the time passed between alarms, Figure 4.5 illustrates, through a dispersion graph, its relation with the expected time between probes (delay used). For visualization reasons, the graph windows have been limited, since some of the time passed values, especially for smaller cycle lengths, highly surpass the expected. The line simply indicates the identity function and, the closer the dots are to it the better, as it signifies anomaly detection time matching the probe delay. Smaller cyclic buffer sizes lead to a greater concentration of data points scattered away from the expected line, indicating deviations from the time expected and suggesting false positives because shorter cycles cannot consistently capture every active service.

The port scan period tends to belong to the left of the identity line because, between probes, if some active service happens to not be present in a cycle, a subsequent alarm may be triggered due to the change in variation whenever it becomes present again. The same logic applies to the period without port

scan. However, since there is no guaranteed probe that triggers it every few seconds, the time passed is irregular and thus the activity may happen at any point. This is why, unless collisions are present, most of the remaining activity, namely the one on the right side of the line, belongs to the period without port scan.

As for the larger cycle lengths, namely 550 and 10000, there is almost no difference between them in terms of dispersion. Considering that an increase in the cyclic buffer size translates to a reduced chance of missing a service and looking at the expected cycle's duration in Table 4.4, it is possible to deduce there were no cycles where services went below five (besides the initial one), meaning that no active services were missed by cycles. Therefore, all the generated alarms are associated with port scan probes which is why most of its values are concentrated on both the identity line and on the double of the time expected.

To verify these claims, analyzing the distribution of the number of occurrences of the different times passed between alarms is essential. As such, Figure 4.6 exemplifies smaller cycle lengths in relation to a cyclic buffer size of 400 and a probe delay of 20 seconds.

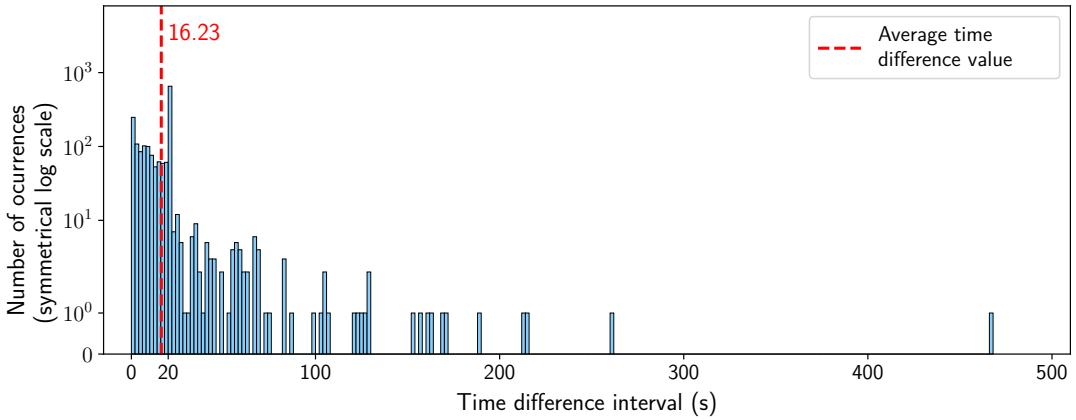


Figure 4.6: Distribution of the number of occurrences of the time passed between alarms - 400 size cyclic buffer & 20 seconds delay

For better visualization, the different times were organized in bins with two seconds of length and a symmetrical logarithmic scale was applied to the number of occurrences. Moreover, the time passed between the first and the second alarms was excluded due to the former being a mandatory alarm, meaning that the time until a second alarm may be too large for an adequate graphical representation.

As expected, given its dispersion, there is a very high variety and distribution of bins, which makes it more challenging to correlate these anomalies with port scan. Considering the cyclic buffer size and looking at the average value, most cycles struggle to contain the fingerprints of the four services leading to false positive alarms between port scan probes, which is why most values are focused on a period below the probe scan delay. Despite that, the most prominent peak, significantly higher than all others

on a logarithmic scale, is concentrated around the 20-second interval, with 654 occurrences in the 20 to 22-second bin.

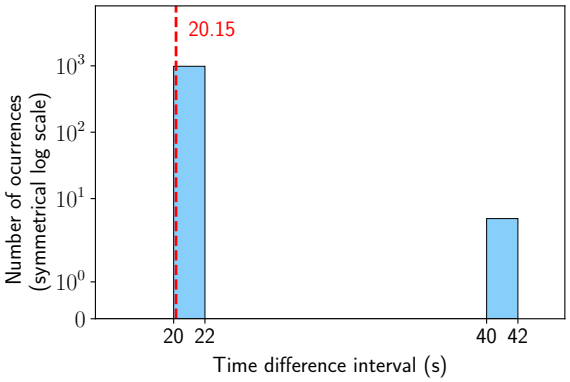


Figure 4.7: Distribution of the number of occurrences of the time passed between alarms - 550 size cyclic buffer & 20 seconds delay

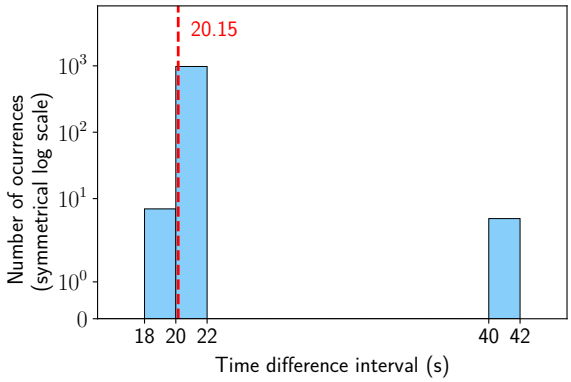


Figure 4.8: Distribution of the number of occurrences of the time passed between alarms - 10000 size cyclic buffer & 20 seconds delay

Concerning the larger cyclic lengths, 550 and 10000, in the Figure 4.5, the difference between them is very small to be noticed. However, the same does not apply when looking at the Figures 4.7 and 4.8, built on the same basis as Figure 4.6, representing the 20-second delay test for the cycle lengths of 550 and 10000 respectively. As expected in the context of a 20-second delay port scan, most values tend to be concentrated around that value. However, in the 10000 cycle length, there is a slightly larger spread of values. Due to its larger size, it is more susceptible to changes in the number of packets per second, which affects the time to complete a cycle, and thus slightly changes the interval between alarms. As for the length 550, the cyclic buffer is completed much more frequently, thus even if the number of packets per second changes, the values will not stray much farther from the original value.

Additionally, there are five values placed within the bin of 40, which is twice as much as the probe delay used. Being this exact value most likely means that alarms are being skipped, which indicates that it is likely a collision with a position of the services at use. Due to the size of the frequency table used, some collisions may occur, especially considering how close the number of probes is to it. To explore such a hypothesis, the Table 4.5 refers to some statistics collected related to Figures 4.7 and 4.8.

Table 4.5: Statistics on the distribution of the number of occurrences of the time passed between alarms - 550 and 10000 size cyclic buffer & 20 seconds delay

Cyclic buffer sizes	Bin Range	Frequency	Average	Standard Deviation	Minimum	Maximum
550	20.0 to 22.0	985	20.05041	0.00903	20.02502	20.09103
	40.0 to 42.0	5	40.10218	0.01418	40.08504	40.12797
10000	18.0 to 20.0	7	19.93499	0.00097	19.93282	19.93591
	20.0 to 22.0	978	40.10218	0.02054	20.04806	20.16669
	40.0 to 42.0	5	40.10218	0.00120	40.09727	40.10054

In Figure 4.3, both cyclic buffer sizes 550 and 10000 have displayed a total of 992 alarms. Excluding the first alarm, that leaves 991, which translates to 990 intervals between alarms, as in the Table 4.5. By adding the five values from the bins 40 to 42 to the 991 alarms, the total would result in 996, which would be two short of the 998 packets related to port scan. However, the two extra packets from the open port (`ACK` and `RST/ACK`) are being sent right away, without having the same delay as the other probes. Therefore, they are likely to occur within the same cycle of the buffer and be mapped to the same position since they refer to the same port, thus not causing an alarm to trigger. To corroborate this, an additional test was performed with a frequency table of size 10000, where the alarm count consistently reached 996, confirming that these were indeed due to collisions.

In the end, the time passed between alarms verifies once again that the anomalies are indeed related to the port scan probes. Moreover, with adequate parameters, P4-CAVODE is successfully capable of detecting 991 anomalies related to the 996 ports scanned, which can be further increased if the frequency table is increased. However, such an increase is not necessary, given that the current number of detected anomalies is likely more than enough for a Controller or an administrator to identify it as a port scan. In what concerns identifying a slow port scan, the time elapsed between alarms can also be an important observation for a Controller, helping to detect deviations in alarm patterns and regularity.

Fast port scan scenario

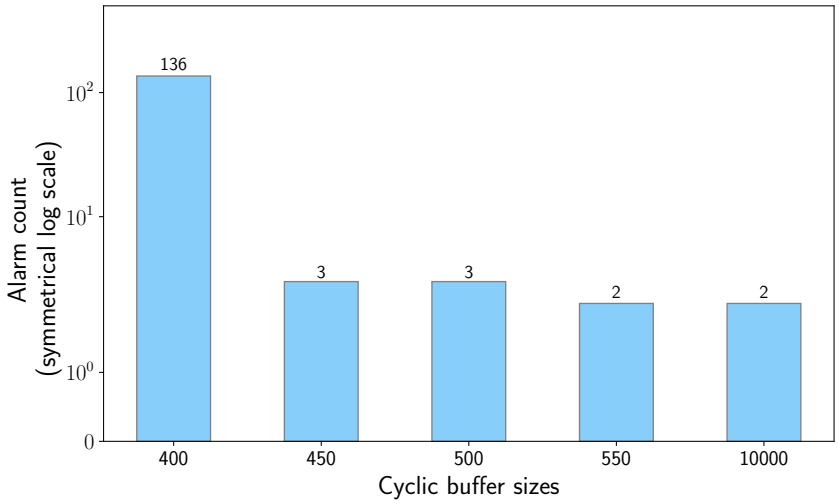


Figure 4.9: Number of alarms per cyclic buffer size

Figure 4.9 shows the number of alarms generated in each cyclic buffer used in the fast scan test described in Section 4.1. At first glance, it becomes clear that there are way fewer alarms when compared to the slow port scan tests. This is because in a fast port scan, different ports are scanned in a very short time span, so its occurrence takes place over fewer cycles when compared to slow scan. As it can be observed, the size 400 generates far more alarms compared to other scans. However, this is most likely the cycle not being able to include every service of the topology which leads to an occasional alarm whenever the distinct counter increases again. This would mean that at some alarms, the distinct counter associated with them would be equal or smaller than four, and, in a fast scan, noticeable bursts in the values held by the distinct counters are expected. To verify this, the distinct counter values associated with each alarm were filtered and listed, so that they are less than four, which leads to the values present in the Table 4.6.

Table 4.6: Distinct counts above 4

Cycle length	Distinct counts
400	23, 219, 270
450	103, 255
500	94, 257
550	245
10000	638

As expected, despite triggering far fewer alarms compared to slow scanning, since the probes are more concentrated, the distinct counter values are much higher, making it far more noticeable. The obtained values for each cycle length happen consecutively and some cycle lengths have more alarms

than others. This happens because the initial alarm triggered by the burst has a value high enough to increase the variation to a point where subsequent distinct counter values are not enough to trigger further alarms. Examples of this are the sizes 400, 450 and 500 where their initial counters are far smaller than the subsequent ones, which keeps causing the variation to trigger alarms. In the last case, the cyclic buffer of size 10000, the value is far larger than any of the present in other sizes. Given that it also covers far more packets in comparison and the fast scan in question only sends probes to 996 ports, the entire scan activity is most likely present in the same cycle.

To corroborate this, two additional tests were conducted, focusing exclusively on fast port scans and excluding any normal activity. Both used a cyclic buffer size of 10000 to include the entire scan activity in a cycle, with the first test employing a frequency table of 1000 and the second using a table of 10000 to observe the difference caused by collisions in the distinct counter. Out of the 996 ports probed, the first test registered a distinct counter of 637, meaning that due to collisions that caused the mapping to the same positions in the frequency table, the estimated number of distinct entries was smaller than the real one. Regarding the second test, it accounted for the total number of ports probed, 996, which confirms that a larger table size increases the likelihood of including every service occurring in a cycle. With more positions available, the hash function has a reduced chance of mapping distinct keys to the same position (collisions), resulting in a more accurate approximation of the distinct count relative to the actual number of distinct entries within each cycle.

Nevertheless, these results only show that the developed solution resourcing to an estimated value suffices, since, even in a fast scan scenario where a higher combination of keys (destination IP and destination port) is expected, the variation always becomes significant enough so that an anomaly may be detected, consequently triggering an alarm. Because these values are more prominent and included in the alarm message, a Controller analyzing the data could readily identify it as a fast port scan.

Sporadic service scenario - slow port scan

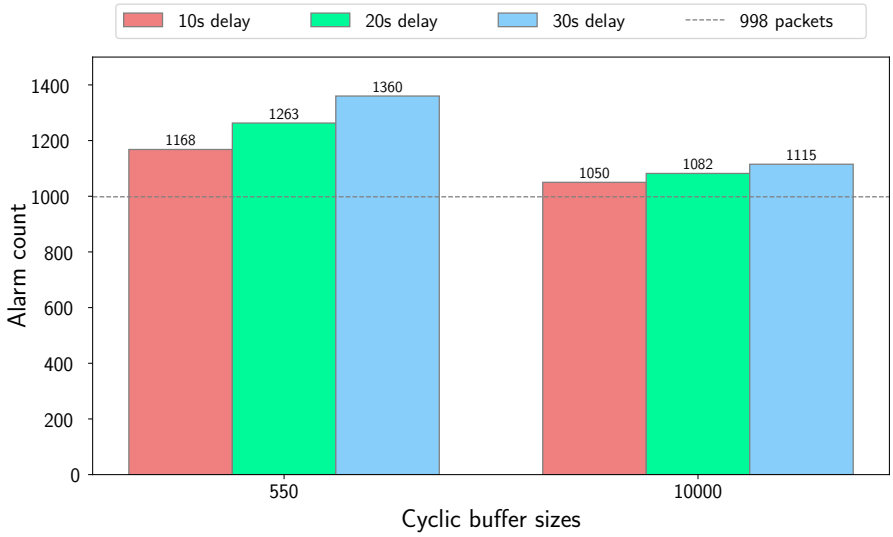


Figure 4.10: Number of alarms per cyclic buffer size - sporadic tests

Bearing in mind that each test is generated individually, there is a certain volatility in the results of smaller sizes, which makes it more challenging for a direct comparison between this scenario and the others. As such, the analysis will focus on the more stable cyclic buffer sizes, namely 550 and 10000. The number of alarms per cyclic buffer size used in the sporadic tests is represented by Figure 4.10. Similarly to the case in the slow scan scenario, there is a reference line that indicates the number of packets relative to the port scan probes. In both cycle lengths, the number of alarms increases alongside the delay. This is because, as mentioned in Section 4.1, the period of activity lasts longer the higher the delay is, which translates to more sporadic service activity that causes false positive alarms to trigger given that the sporadic service is not present in every cycle. Considering the duration of the test scenario described in Section 4.1, the average number of packets per second in Figure 4.2, and that the sporadic service performs ten seconds of activity every five minutes, that would lead to the approximate total number of packets displayed in Table 4.7.

Table 4.7: Approximate total number of sporadic service packets per test

Delay used (s)	Test duration	Approximate total number of packets
No delay used	8000	189800
10	18000	438000
20	28000	678900
30	38000	919800

Despite the large amount of activity performed by the sporadic service, thanks to P4-CAVODE's adaptative properties regarding the variation that controls the alarms generated, that only translates to a very small percentage of alarms in comparison.

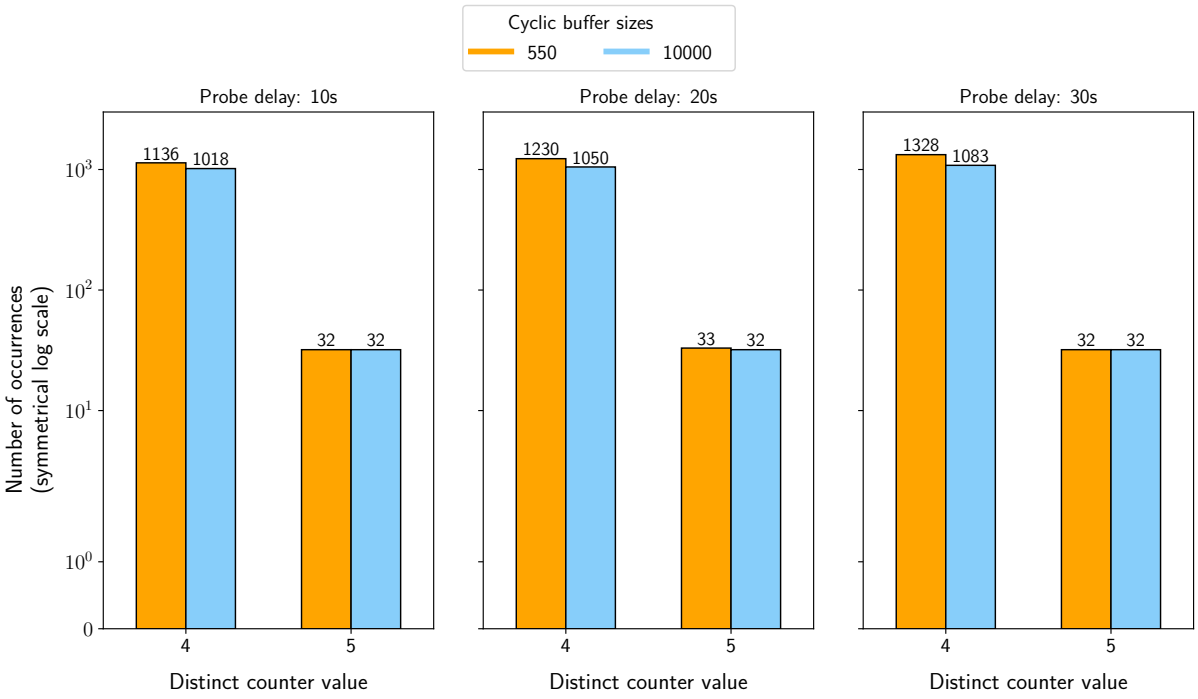


Figure 4.11: Distribution of distinct counter values for different cyclic buffer sizes and delays

Figure 4.11 shows the distribution of the distinct counter values associated with each alarm. Since in this scenario only three services are continuously active, if every cycle includes those three services, then alarms should trigger when the distinct counter goes above three. None of the cyclic buffer sizes triggered alarms with distinct counters below four, indicating that an alarm trigger only occurred when the fourth service or scan activity was present. As mentioned earlier in the slow scan scenario, thanks to how often cycles are updated, there can only be at most one slow port scan probe per cycle, meaning that the values of five represent when both the fourth service and scan activity were simultaneously present.

The smaller cycle length, 550, featured more fingerprints at four than the 10000, with the latter being closer to the reference line in Figure 4.10. To clarify the results, additional tests were conducted with all services initially active, where the sporadic activity ceased after ten seconds and then resumed for another ten seconds a minute later. Excluding the first alarm, overall most cyclic buffer sizes consistently resulted in four alarms, with only the largest, 10000, producing a single alarm. After analyzing the alarms, it was found that the second alarm occurred about ten seconds after the initial alarm, near the end of the sporadic activity. Then, a minute later, three additional alarms were recorded: one at the start of the sporadic activity, another very shortly after, and a final one once again near the end of the activity. Upon

inspection of the packet capture, perhaps due to how the *iPerf3* works for not being a continuous activity as the other services, there are very small pauses in the traffic at both the beginning and before the end of the activity. These pauses, combined with the activity of other services, result in cycles without the sporadic service, causing an alarm to trigger each time the sporadic service packet is resent. An increase in the buffer, as in the case of the size 10000, heavily mitigates this situation by containing many more packets and extending the duration of each cycle, therefore surpassing the said pauses. Nonetheless, the number of generated alarms is still very small in comparison to the number of packets produced.

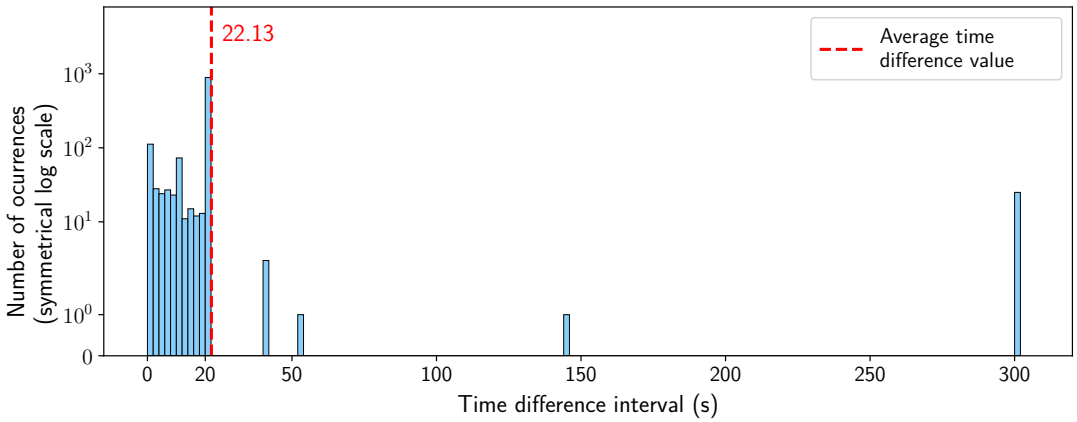


Figure 4.12: Distribution of the number of occurrences of the time passed between alarms - 550 size cyclic buffer & 20 seconds delay

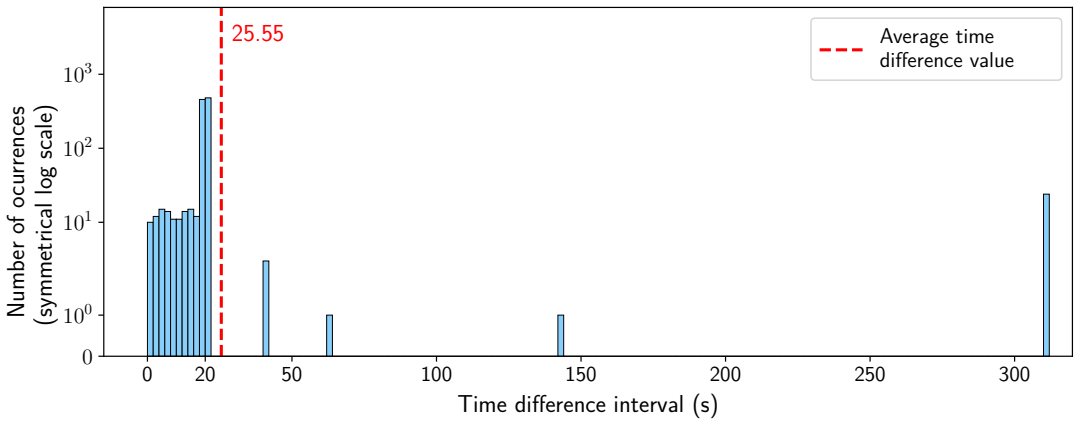


Figure 4.13: Distribution of the number of occurrences of the time passed between alarms - 10000 size cyclic buffer & 20 seconds delay

Figures 4.12 and 4.13 represent the distribution of the number of the number of occurrences of the intervals of time passed between alarms for both sizes under analysis, using a probe delay of 20 seconds. Since the service occurs every five minutes, this means that at least every 300 seconds the number of

keys will increase by at least one. Outside the port scan period, this will cause false positive alarms, which justifies its frequency at the 300 seconds (five minutes) mark.

The distribution of intervals is considerably wider compared to the slow scan scenario, with a concentration of intervals falling below the probe delay threshold, which could make it harder to correlate with a scan at first glance. Service probes may coincide with or even fall between scan probes, which could affect variation or even trigger alarms. In turn, these changes in the variation could trigger additional alarms until it adapts. Adjusting the cyclic buffer size and even variation factors such as α and γ could provide improvements on this front. Despite that, considering that the number of occurrences is on a logarithmic scale, the peak around the 20-seconds bin itself is still significantly higher than any other frequency (symmetrical logarithmic scale), which can be considered suspicious when observed by an entity such as a Controller or administrator. Since the control plane receives information regarding what triggers an alarm, the use of a Controller could prove beneficial in the sense that it can have visibility over multiple data planes, enabling a more efficient identification and correlation of patterns. For instance, in the context of sporadic services, a Controller could hold information about the periodicity of the access of such services and combine it with an implementation of an anomaly detection solution with the time passed between the alarms as the subject.

Sporadic service scenario - fast port scan

As for the fast port scan test, the number of alarms obtained per cycle buffer is as follows, in Figure 4.14.

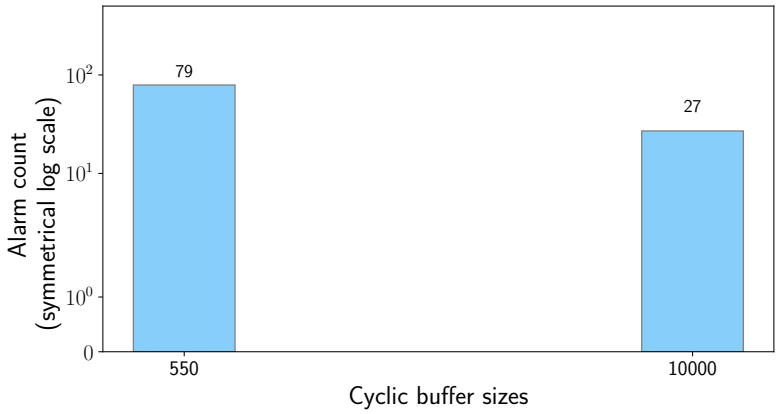


Figure 4.14: Number of alarms per cyclic buffer size

The total number of alarms is much smaller than the slow scan one. This is because it occurs over a much smaller timeframe, causing the port scan activity to be concentrated within a few consecutive cycles. Once again, the larger cyclic buffer results in fewer alarms due to its ability to hold more packets, which manages to include the sporadic service's distinct behavior. Since the traffic of the sporadic service is

only sent every five minutes, the initial period does not include it. Thus, similar to the slow scan scenario of the sporadic service, three alarms are expected for each period of sporadic activity with a cyclic buffer size of 550, and one alarm with a cyclic buffer size of 10000. Excluding the initial mandatory alarm, this expectation aligns with the observed alarm counts: 78 for the 550 buffer size and 26 for the 10,000 buffer size, reflecting a precise three-to-one ratio. Under the same assumptions as the fast scan scenario, an abrupt change in terms of the distinct counter is to be expected when in comparison to the rest of the alarms. Therefore, applying the same filter to include only entries above four results in Table 4.8.

Table 4.8: Distinct counts above 4

Cycle length	Distinct counts
550	59, 318
10000	537

Since the scan activity is more concentrated, the values of the distinct counter and consequently the variation are much higher than the ones pertaining to the sporadic service, which makes it more straightforward to distinguish when compared to the slow scan test. From a Controller perspective, in addition to what is mentioned in the slow port scan test of the sporadic service scenario, distinguishing this type of port scan from the sporadic service could be based on the number of distinct entries indicated by the alarm.

4.2.3 Throughput

Regarding the throughput scenario described in Section 4.1, in the case of the single service tests, the cycle is entirely composed of the same fingerprint, meaning that the fingerprint that is to be replaced will always coincide with the new fingerprint. According to the description in Section 3.1, the entire frequency table process and the update on the distinct counter is only entered if both fingerprints differ, as follows:

```
if (old_fingerprint != new_fingerprint) {
    process_table();
  }
```

This is because when dealing with identical fingerprints, the frequency changes cancel each other, leading to no alterations of the distinct counter, which makes these operations irrelevant. While a scenario with a single service test benefits the most from this optimization, a system with a fixed amount of services, such as the scenario with four services, will still take advantage of it since in the cycles of the buffer there will still be occasions where the said fingerprints will be the same. To verify this, three additional one-

minute tests were conducted on the Netronome SmartNIC under the same conditions with four services, incorporating an extra counter to track the number of occurrences of the removed fingerprint matching its replacement during each cycle. In a total of around 48 million packets processed, the counters recorded approximately 38 million, 15 million, and 24 million matches in each test. Due to the probabilistic nature of this matching condition, repeated tests yield different counter values. Nevertheless, they demonstrate the effectiveness of the optimization that allows to bypass read and write operations related to the frequency table and the distinct counter.

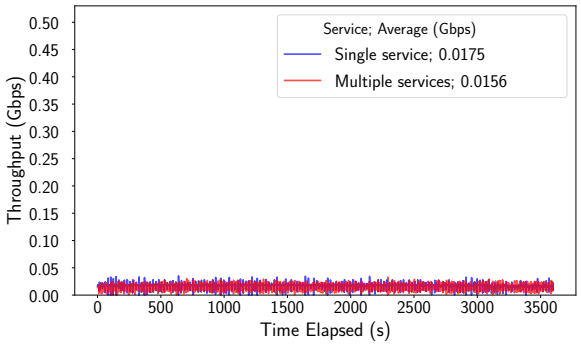


Figure 4.15: Achieved throughput and its scalability on BMv2

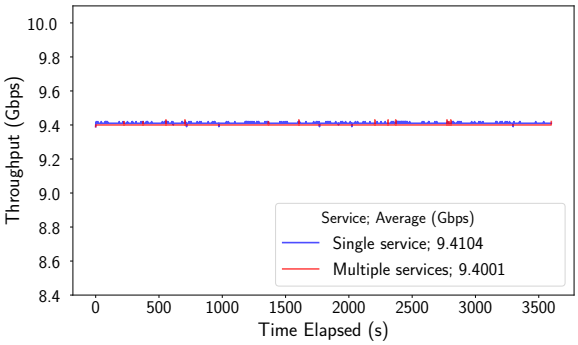


Figure 4.16: Achieved throughput and its scalability on Netronome SmartNIC

The Figures 4.15 and 4.16 display the throughput scenario described in Section 4.1. The red line pertains to only one service being used and the blue line to the four services of the topology in Figure 4.1. On both targets, thanks to the described condition, the table process is always skipped, hence why the single service displays a slightly higher throughput in those situations. Now, regarding the actual achieved values, BMv2 shows significantly inferior values, averaging 17.5 Mbps on the single service test and 15.6 Mbps on the multiple services test. This is because it only has a single thread in the ingress pipeline, which means it can only process a single packet at a time and therefore it should only be used in proof-of-concept applications, as it is not suitable for real environments.

On the other hand, the Netronome SmartNIC possesses multiple threads and thus can process multiple packets simultaneously, meaning that it can benefit from lock granularity. As addressed in Section 3.2.3, the lock granularity isolates operations of the different structures, even the more complex ones, such as the cyclic buffer and frequency table, by applying locks to each entry, allowing multiple packets to be processed simultaneously, thus mitigating possible congestions in the system. As such, in the hardware implementation, P4-CAVODE manages to achieve a very high performance by maintaining near-line-rate processing on an interface with a theoretical maximum of 10Gbps, averaging around 9.41 Gbps on the

single service test and 9.40 Gbps on the multiple services test. The reason why the single service slightly surpasses the multiple services is due to ignoring an entire set of operations, namely the table process, as described. Despite that, thanks to the described optimizations, namely the lock granularity, the system is still able to achieve outstanding throughput values on the multiple services test by taking advantage of the Netronome SmartNIC characteristics, namely the multiple threads of its FPCs, which are focused on packet and flow processing (see Section 2.1). The scalability in terms of services and the overall optimal performance shown through these results highlight the viability of employing, in the data plane, tasks that once were the responsibility of the control plane.

4.3 Summary

This chapter started by describing the test scenarios and the considerations behind them. It then proceeded to present the said tests under an analysis based on critical parameters such as accuracy, detection latency and throughput. Regarding the slow scan scenario, the accuracy of the P4-CAVODE increases the higher the cyclic buffer size is, since it is capable of containing more packets, thus increasing the odds of all the services being included in the cycle. Even when producing an average of 2936 packets per second, using a cyclic buffer of size 550 suffices to show the accuracy of the system, which can successfully detect 991 anomalies out of the 996 ports scanned. While this can be further increased by increasing the frequency table, the trade-off in terms of memory is not worth it, as a Controller or administrator should already be able to link these anomalies to the occurrence of port scan. Additionally, a larger cyclic buffer size means that each cycle takes longer to complete, resulting in higher anomaly detection latency. However, even with a drastic increase in cyclic buffer size from 550 to 10000, the detection delay did not increase significantly enough to prevent timely countermeasures against subsequent attacks. As for fast scans, their aggressive nature causes abrupt changes in the distinct counter and consequently, the variation, which makes it more straightforward for a system to determine if their respective anomalies could be a port scan occurrence. The scenario pertaining to a sporadic service showed more challenging results in the sense that it might be harder to distinguish from a slow scan occurrence. However, a controller with a higher view of the system could implement solutions to discern this behavior. Despite this, the results also demonstrated the P4-CAVODE's adaptability, as thousands of packets were reduced to only tens to hundreds of alarms, thanks to its ability to adapt to variations. In the sporadic service scenario with a fast port scan, the detection results were similar to the fast scan scenario, as the alarms' distinct counters were significantly higher than those in the other alarms. Lastly, the throughput achieved on both tests with a single service and with multiple services was near the theoretical value of 10Gbps,

showing that offloading an anomaly detection solution such as P4-CAVODE does not hinder the system's normal functioning. Overall, the results demonstrated that the system, using structures that store minimal information, was capable of detecting anomalies caused by port scans with different behaviors. This highlights the feasibility of deploying anomaly detection solutions in the data plane without restricting network throughput.

Chapter 5
Conclusions

The objectives outlined in Section 1.2 aimed to demonstrate the feasibility of implementing cybersecurity solutions directly within SDNs, specifically through the use of PDPs. The choice for this was the development of an anomaly detection system called P4-CAVODE, where its object of detection consisted of the *reconaissance* technique called port scan, namely the slow port scan variant.

Bearing in mind the limitations of the data plane in terms of memory and the need to keep track of data and statistics, P4-CAVODE resources to structures with reduced memory footprint known as membership and counting structures. The detection of anomalies is done through the usage of algorithms that detect deviations in data streams. Despite the object of detection leaning towards the detection of anomalies caused by slow port scans, the algorithm is also capable of detecting fast port scans. Moreover, the format of the keys used in the membership structures ensures that the algorithm is independent of header strategies such as the ones mentioned in Section 2.3.2 and is also capable of handling target multiplicity. In Section 4.2, P4-CAVODE is shown to be capable of successfully identifying anomalies that can be associated with the probes of the said port scans.

Despite the detection of anomalies being based on the behavior of port scans, changing the subject of observation to fit the characteristics of other types of techniques applied in the stages that lead to a cyberattack or even the attack itself may allow for their detection. Besides, even without explicitly adapting the solution, structures and mechanisms similar to the ones used are often employed in the field of intrusion detection even if they only serve as components that offload tasks from a larger system.

P4-CAVODE features a software implementation as proof-of-concept and another implementation that is deployed in hardware. While the portability proved to be a challenge due to a hardware compiler-related error, the remaining aspects of the transition from software to hardware targets were more straightforward thanks to the restrictions and constructs of the P4 language, with the required changes being related to specific hardware characteristics. Adding those points to the capability of being able to simply modify the P4 program at will and redeploy it directly onto the hardware highlights the importance of data plane programmability in the field of network security, where the discussed features are essential.

Through the optimizations performed, namely the implementation of lock granularity on the memory operations logic which allowed to leverage the hardware capabilities, particularly the FPCs, P4-CAVODE is able to handle multiple services while maintaining near-theoretical line-rate performance. These characteristics affirm its scalability and effectively demonstrate the feasibility of enhancing cybersecurity within network devices.

5.1 Contributions

The main contribution of this research lies in the proposed anomaly detection system itself, P4-CAVODE. While anomaly detection solutions utilizing software targets like BMv2 are minimally well-established, the same cannot be said for those related to hardware targets. The significance of our work lies in demonstrating the feasibility of employing Programmable Data Planes within hardware targets, particularly in the field of cybersecurity.

Ultimately, this approach has resulted in a scalable solution through fine lock granularity, delivering throughput performance that approaches the actual bandwidth of the hardware. Moreover, while using structures intended for reduced memory usage, it also proved itself capable of detecting anomalies with high accuracy and minimum delay, pointing out its effectiveness and reliability in high-performance network environments.

5.2 Future work

The fundamental concept of the P4-CAVODE's current iteration focuses on highlighting whenever an anomaly is detected. It does not take any actual conclusion whether or not an attack is being performed, thus not being able to distinguish between anomalies and sporadic probes to services (Section 4.2). Future iterations could include mechanisms capable of making such decisions, such as a Controller, with logic implemented. Also, since the Controller can have visibility over multiple dataplanes, this leads to many possibilities such as correlating alarms between them, which could increase the detection scope. However, improvements are not exclusive to the notion of a Controller. At the cost of a larger memory footprint, the data plane itself can be improved to identify the source of attacks by keeping track of source IPs as in solutions mentioned in Sections 2.5 and 2.8.

Regarding the performance factor, even though P4-CAVODE already approaches the theoretical line-rate of the hardware, the distinct counter is being read even when there is no cyclic buffer reset, as indicated in Section 3.2. While this method outperforms the alternative of including it in the reset condi-

tion, it still leads to N-1 redundant read operations per cycle (N represents the size of the cyclic buffer). Therefore, performing a detailed analysis of the intermediate code produced by the compiler could provide valuable insights into further optimizing the system by eliminating unnecessary operations such as the one described and enhancing its overall performance.

Bibliography

[1] E. S. Page. “Cumulative Sum Charts”. In: *Technometrics* 3.1 (Feb. 1961), pp. 1–9. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1961.10489922. URL: http://www.tandfonline.com/doi/abs/10.1080/00401706.1961.10489922 (visited on 09/22/2023).

[2] Burton H. Bloom. “Space/time trade-offs in hash coding with allowable errors”. In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/362686.362692. URL: https://dl.acm.org/doi/10.1145/362686.362692 (visited on 09/10/2023).

[3] *Transmission Control Protocol*. Request for Comments RFC 793. Num Pages: 91. Internet Engineering Task Force, Sept. 1981. DOI: 10.17487/RFC0793. URL: https://datatracker.ietf.org/doc/rfc793 (visited on 01/14/2023).

[4] James M. Lucas. “Cumulative sum (cusum) control schemes”. In: *Communications in Statistics - Theory and Methods* 14.11 (Jan. 1985), pp. 2689–2704. ISSN: 0361-0926, 1532-415X. DOI: 10.1080/03610928508829070. URL: http://www.tandfonline.com/doi/abs/10.1080/03610928508829070 (visited on 09/22/2023).

[5] Urupoj Kanlayasiri, Surasak Sanguanpong, and Wipa Jaratmanachot. “A rule-based approach for port scanning detection”. In: *Proceedings of the 23rd electrical engineering conference, Chiang Mai Thailand*. Citeseer. 2000, pp. 485–488.

[6] Cynthia Bailey Lee, Chris Roedel, and Elena Silenok. “Detection and characterization of port scan attacks”. In: *Univeristy of California, Department of Computer Science and Engineering* (2003).

[7] Ramana Rao Kompella, Sumeet Singh, and George Varghese. “On scalable attack detection in the network”. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. IMC ’04. New York, NY, USA: Association for Computing Machinery, Oct. 2004, pp. 187–200. ISBN: 978-1-58113-821-4. DOI: 10.1145/1028788.1028812. URL: https://doi.org/10.1145/1028788.1028812 (visited on 12/04/2022).

[8] Graham Cormode and S. Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications”. In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0196677403001913.

[9] S. Panjwani et al. “An experimental evaluation to determine if port scans are precursors to an attack”. In: *2005 International Conference on Dependable Systems and Networks (DSN’05)*. ISSN: 2158-3927. June 2005, pp. 602–611. DOI: 10.1109/DSN.2005.18.

[10] Tarek S. Sobh. “Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art”. In: *Computer Standards & Interfaces* 28.6 (Sept. 2006), pp. 670–694. ISSN: 09205489. DOI: 10.1016/j.csi.2005.07.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S092054890500098X (visited on 01/15/2023).

[11] Albert Bifet and Ricard Gavaldà. “Learning from Time-Changing Data with Adaptive Windowing”. In: *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, Apr. 2007, pp. 443–448. ISBN: 978-0-89871-630-6. DOI: 10.1137/1.9781611972771.42. URL: https://epubs.siam.org/doi/10.1137/1.9781611972771.42 (visited on 06/12/2023).

[12] Steven Cheung et al. “Using model-based intrusion detection for SCADA networks”. In: *Proceedings of the SCADA security scientific symposium*. Vol. 46. SRI International. 2007, pp. 1–12.

[13] K A Scarfone and P M Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. Tech. rep. NIST SP 800-94. Edition: 0. Gaithersburg, MD: National Institute of Standards and Technology, 2007, NIST SP 800–94. DOI: 10.6028/NIST.SP.800-94. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf (visited on 01/08/2023).

[14] Raquel Sebastião and João Gama. “Change Detection in Learning Histograms from Data Streams”. In: *Progress in Artificial Intelligence*. Ed. by José Neves, Manuel Filipe Santos, and José Manuel Machado. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 112–123. ISBN: 978-3-540-77002-2. DOI: 10.1007/978-3-540-77002-2_10.

[15] Jayant Gadge and Anish Anand Patil. “Port scan detection”. In: *2008 16th IEEE International Conference on Networks*. ISSN: 2332-5798. Dec. 2008, pp. 1–6. DOI: 10.1109/ICON.2008.4772622.

[16] Raquel Sebastiao and Joao Gama. “A study on change detection methods”. In: *Progress in artificial intelligence, 14th Portuguese conference on artificial intelligence, EPIA*. 2009, pp. 12–15.

[17] Himanshu Singh. “Distributed Port Scanning Detection”. Master of Science. San Jose, CA, USA: San Jose State University, Jan. 2009. DOI: 10.31979/etd.k3gy-z9up. URL: https://scholarworks.sjsu.edu/etd_projects/142 (visited on 11/22/2022).

[18] Raquel Sebastião et al. “Monitoring Incremental Histogram Distribution for Change Detection in Data Streams”. In: *Knowledge Discovery from Sensor Data*. Ed. by David Hutchison et al. Vol. 5840. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 25–42. ISBN: 978-3-642-12518-8 978-3-642-12519-5. DOI: 10.1007/978-3-642-12519-5_2. URL: http://link.springer.com/10.1007/978-3-642-12519-5_2 (visited on 06/13/2023).

[19] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. “Surveying Port Scans and Their Detection Methodologies”. In: *The Computer Journal* 54.10 (Oct. 2011), pp. 1565–1581. ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxr035. URL: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/bxr035 (visited on 11/16/2022).

[20] Michelle Cotton et al. *Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry*. Tech. rep. 2011.

[21] Mehiar Dabbagh et al. “Slow port scanning detection”. In: *2011 7th International Conference on Information Assurance and Security (IAS)*. Dec. 2011, pp. 228–233. DOI: 10.1109/ISIAS.2011.6122824.

[22] Michael A. Bender et al. “Don’t thrash: how to cache your hash on flash”. In: *Proceedings of the VLDB Endowment* 5.11 (July 2012), pp. 1627–1637. ISSN: 2150-8097. DOI: 10.14778/2350229.2350275. URL: https://dl.acm.org/doi/10.14778/2350229.2350275 (visited on 09/10/2023).

[23] H. P.Sanghvi and M. S. Dahiya. “Cyber Reconnaissance: An Alarm before Cyber Attack”. In: *International Journal of Computer Applications* 63.6 (Feb. 2013), pp. 36–38. ISSN: 09758887. DOI: 10.5120/10472-5202. URL: http://research.ijcaonline.org/volume63/number6/pxc3885202.pdf (visited on 02/07/2023).

[24] Timon Sloane. *OpenFlow*. Feb. 2013. URL: https://opennetworking.org/sdn-resources/customer-case-studies/openflow/ (visited on 02/09/2023).

[25] Pat Bosshart et al. “P4: programming protocol-independent packet processors”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://dl.acm.org/doi/10.1145/2656877.2656890 (visited on 10/06/2022).

[26] Elias Bou-Harb, Mourad Debbabi, and Chadi Assi. “Cyber Scanning: A Comprehensive Survey”. In: *IEEE Communications Surveys & Tutorials* 16.3 (2014). Conference Name: IEEE Communications Surveys & Tutorials, pp. 1496–1519. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.102913.00020.

[27] Yousra Chabchoub, Raja Chiky, and Betul Dogan. “How can sliding HyperLogLog and EWMA detect port scan attacks in IP traffic?” In: *EURASIP Journal on Information Security* 2014.1 (Dec. 2014), p. 5. ISSN: 1687-417X. DOI: 10.1186/1687-417X-2014-5. URL: https://jis-eurasipjournals.springeropen.com/articles/10.1186/1687-417X-2014-5 (visited on 06/09/2023).

[28] Bin Fan et al. “Cuckoo Filter: Practically Better Than Bloom”. In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. Sydney Australia: ACM, Dec. 2014, pp. 75–88. ISBN: 978-1-4503-3279-8. DOI: 10.1145/2674005.2674994. URL: https://dl.acm.org/doi/10.1145/2674005.2674994 (visited on 09/12/2023).

[29] Sumit Kumar and Sithu D. Sudarsan. “An Innovative UDP Port Scanning Technique”. In: *International Journal of Future Computer and Communication* 3.6 (Dec. 2014), pp. 381–384. ISSN: 20103751. DOI: 10.7763/IJFCC.2014.V3.332. URL: https://www.ijfcc.org/papers/332-N0012.pdf (visited on 11/16/2022).

[30] Noa Zilberman et al. “NetFPGA SUME: Toward 100 Gbps as Research Commodity”. In: *IEEE Micro* 34.5 (Sept. 2014). Conference Name: IEEE Micro, pp. 32–41. ISSN: 1937-4143. DOI: 10.1109/MM.2014.61. URL: https://ieeexplore.ieee.org/document/6866035/?arnumber=6866035 (visited on 08/08/2024).

[31] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. arXiv:1511.08458 [cs]. Dec. 2015. URL: http://arxiv.org/abs/1511.08458 (visited on 02/09/2023).

[32] Fu-Hau Hsu et al. “TRAP: A Three-Way Handshake Server for TCP Connection Establishment”. In: *Applied Sciences* 6.11 (Nov. 2016). Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, p. 358. ISSN: 2076-3417. DOI: 10.3390/app6110358. URL: https://www.mdpi.com/2076-3417/6/11/358 (visited on 01/16/2023).

[33] P4.org. *Clarifying the differences between P4 and OpenFlow*. May 2016. URL: https://opennetworking.org/news-and-events/blog/clarifying-the-differences-between-p4-and-openflow/ (visited on 02/09/2023).

[34] William R. Simpson and Kevin E. Foltz. *Enterprise Considerations for Ports and Protocols*. Tech. rep. Institute for Defense Analyses, 2016. URL: https://www.jstor.org/stable/resrep22700 (visited on 12/02/2022).

[35] Evgeny V. Ananin, Arina V. Nikishova, and Irina S. Kozhevnikova. “Port scanning detection based on anomalies”. In: *2017 Dynamics of Systems, Mechanisms and Machines (Dynamics)*. Nov. 2017, pp. 1–5. DOI: 10.1109/Dynamics.2017.8239427.

[36] Markus Ring et al. “Creation of flow-based data sets for intrusion detection”. In: *Journal of Information Warfare* 16.4 (2017), pp. 41–54.

[37] Eran Assaf et al. “Pay for a Sliding Bloom Filter and Get Counting, Distinct Elements, and Entropy for Free”. In: Apr. 2018, pp. 2204–2212. DOI: 10.1109/INFOCOM.2018.8485882.

[38] Gorby Kabasele Ndonda and Ramin Sadre. “A Two-level Intrusion Detection System for Industrial Control System Networks using P4”. In: 2018. URL: https://dial.uclouvain.be/pr/boreal/object/boreal:200310 (visited on 10/07/2022).

[39] “NFP-4000 Theory of Operation”. In: 2018. URL: https://web.archive.org/web/20220730092929/https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf (visited on 04/24/2024).

[40] Suad Mohammed Othman et al. “Survey on intrusion detection system types”. In: *International Journal of Cyber-Security and Digital Forensics* 7.4 (2018), pp. 444–463.

[41] Francesco Paolucci et al. “P4 Edge Node Enabling Stateful Traffic Engineering and Cyber Security”. In: *Journal of Optical Communications and Networking* 11 (Oct. 2018), A84. DOI: 10.1364/JOCN.11.000A84.

[42] Markus Ring, Dieter Landes, and Andreas Hotho. “Detection of slow port scans in flow-based network traffic”. In: *PLOS ONE* 13.9 (Sept. 2018). Publisher: Public Library of Science, e0204507. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0204507. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0204507 (visited on 11/14/2022).

[43] Noa Zilberman. *P4 Tutorial Welcome*. 2018. URL: https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf (visited on 12/23/2022).

[44] Stephen Ibanez et al. “The P4->NetFPGA Workflow for Line-Rate Packet Processing”. en. In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Seaside CA USA: ACM, Feb. 2019, pp. 1–9. ISBN: 978-1-4503-6137-8. DOI: 10.1145/3289602.3293924. URL: https://dl.acm.org/doi/10.1145/3289602.3293924 (visited on 08/08/2024).

[45] Stephen Ibanez et al. “The P4->NetFPGA Workflow for Line-Rate Packet Processing”. In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA ’19. Seaside, CA, USA: Association for Computing Machinery, 2019, pp. 1–9. ISBN: 9781450361378. DOI: 10.1145/3289602.3293924. URL: https://doi.org/10.1145/3289602.3293924.

[46] Jiyeon Kim, Yulim Shin, and Eunjung Choi. “An Intrusion Detection Model based on a Convolutional Neural Network”. In: *Journal of Multimedia Information System* 6.4 (Dec. 2019), pp. 165–172. ISSN: 2383-7632. DOI: 10.33851/JMIS.2019.6.4.165. URL: http://www.jmis.org/archive/view_article?doi=10.33851/JMIS.2019.6.4.165 (visited on 01/09/2023).

[47] Benjamin Lewis, Matthew Broadbent, and Nicholas Race. “P4ID: P4 Enhanced Intrusion Detection”. In: *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Nov. 2019, pp. 1–4. DOI: 10.1109/NFV-SDN47374.2019.9040044.

[48] Markus Ring et al. “A Survey of Network-based Intrusion Detection Data Sets”. In: *Computers & Security* 86 (Sept. 2019). arXiv:1903.02460 [cs], pp. 147–167. ISSN: 01674048. DOI: 10.1016/j.cose.2019.06.005. URL: http://arxiv.org/abs/1903.02460 (visited on 10/12/2023).

[49] Moona Olakara Mohammed. “Automatic Port Scanner”. In: *International Journal of Innovative Science and Research Technology* 5.9 (Sept. 2020), pp. 711–717. ISSN: 24562165. DOI: 10.38124/IJISRT20SEP503. URL: https://ijisrt.com/assets/upload/files/IJISRT20SEP503.pdf (visited on 11/30/2022).

[50] Mehr u Nisa and Kashif Kifayat. “Detection of Slow Port Scanning Attacks”. In: *2020 International Conference on Cyber Warfare and Security (ICCWS)*. Oct. 2020, pp. 1–7. DOI: 10.1109/ICCWS48432.2020.9292389.

[51] Amir Alsadi et al. “A Security Monitoring Architecture based on Data Plane Programmability”. In: *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. ISSN: 2575-4912. June 2021, pp. 389–394. DOI: 10.1109/EuCNC/6GSummit51104.2021.9482549.

[52] Gereltsetseg Altangerel. “Survey on Some Optimization Possibilities for Data Plane Applications”. In: *Machine Learning Techniques and Data Science*. Academy and Industry Research Collaboration Center (AIRCC), Nov. 2021, pp. 69–81. ISBN: 978-1-925953-52-7. DOI: 10.5121/csit.2021.111807. URL: https://aircconline.com/csit/papers/vol11/csit111807.pdf (visited on 02/09/2023).

[53] Lina Blomkvist and Tove Svensson. “Integrating Programmable Smart-NICs into Industrial Packet-Processing Systems”. In: (2021).

[54] Ya Gao and Zhenling Wang. “A Review of P4 Programmable Data Planes for Network Security”. In: *Mobile Information Systems* 2021 (Nov. 2021). Publisher: Hindawi, e1257046. ISSN: 1574-017X. DOI: 10.1155/2021/1257046. URL: https://www.hindawi.com/journals/misy/2021/1257046/ (visited on 11/16/2022).

[55] Nicholas Gray et al. “High Performance Network Metadata Extraction Using P4 for ML-based Intrusion Detection Systems”. In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. ISSN: 2325-5609. June 2021, pp. 1–7. DOI: 10.1109/HPSR52026.2021.9481849.

[56] Saad Haji et al. “Comparison of Software Defined Networking with Traditional Networking”. In: *Asian Journal of Computer Science and Information Technology* 9 (May 2021), pp. 1–18. DOI: 10.9734/AJRCOS/2021/v9i230216.

[57] Hasanin Harkous et al. “Performance Study of P4 Programmable Devices: Flow Scalability and Rule Update Responsiveness”. In: *2021 IFIP Networking Conference (IFIP Networking)*. ISSN: 1861-2288. June 2021, pp. 1–6. DOI: 10.23919/IFIPNetworking52078.2021.9472782. URL: https://ieeexplore.ieee.org/document/9472782/?arnumber=9472782 (visited on 08/08/2024).

[58] Hasanin Harkous et al. “Virtual Queues for P4: A Poor Man’s Programmable Traffic Manager”. In: *IEEE Transactions on Network and Service Management* 18.3 (Sept. 2021), pp. 2860–2872. ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2021.3077051. URL: https://ieeexplore.ieee.org/document/9420725/ (visited on 04/21/2024).

[59] A. S. Ilha et al. “Euclid: A Fully In-Network, P4-Based Approach for Real-Time DDoS Attack Detection and Mitigation”. In: *IEEE Transactions on Network and Service Management* 18.3 (Sept. 2021), pp. 3121–3139. DOI: 10.1109/TNSM.2020.3048265.

[60] Sukhveer Kaur, Krishan Kumar, and Naveen Aggarwal. “A review on P4-Programmable data planes: Architecture, research efforts, and future directions”. In: *Computer Communications* 170 (Mar. 2021), pp. 109–129. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2021.01.027. URL: https://www.sciencedirect.com/science/article/pii/S0140366421000487 (visited on 10/06/2022).

[61] Elie F. Kfoury, Jorge Crichigno, and Elias Bou-Harb. “An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends”. In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 87094–87155. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3086704.

[62] Wojciech Mazurczyk and Luca Caviglione. “Cyber reconnaissance techniques”. In: *Communications of the ACM* 64.3 (Mar. 2021), pp. 86–95. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3418293. URL: https://dl.acm.org/doi/10.1145/3418293 (visited on 02/08/2023).

[63] Merve Ozkan-Okay et al. “A Comprehensive Systematic Literature Review on Intrusion Detection Systems”. In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 157727–157760. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3129336.

[64] Kairo Tavares and Tiago Ferreto. “P4-ONIDS: A P4-based NIDS optimized for constrained programmable data planes in SDN”. In: *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2021)*. Brasil: Sociedade Brasileira de Computação - SBC, Aug. 2021, pp. 434–447. DOI: 10.5753/sbrc.2021.16738. URL: https://sol.sbc.org.br/index.php/sbrc/article/view/16738 (visited on 10/14/2022).

[65] Irina Tsareva, Dominik Scholz, and Sebastian Gallenmüller. “Taxonomy of the Performance of P4 Targets”. In: *Network* 25 (2021).

[66] Gergő Gombos et al. “Active Queue Management on the Tofino programmable switch: The (Dual)PI2 case”. In: *ICC 2022 - IEEE International Conference on Communications*. 2022, pp. 1685–1691. DOI: 10.1109/ICC45855.2022.9838674.

[67] Athanasios Liatifis et al. “Advancing SDN: from OpenFlow to P4, a Survey”. In: *ACM Computing Surveys* (Aug. 2022). Just Accepted. ISSN: 0360-0300. DOI: 10.1145/3556973. URL: https://doi.org/10.1145/3556973 (visited on 10/07/2022).

[68] *$P4_{16}$ Language Specification*. 2022. URL: https://p4.org/p4-spec/docs/P4-16-v1.2.3.pdf (visited on 12/23/2022).

[69] *Cyber Kill Chain®*. Feb. 2023. URL: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html (visited on 02/08/2023).

[70] Frederik Hauser et al. “A survey on data plane programming with p4: Fundamentals, advances, and applied research”. In: *Journal of Network and Computer Applications* 212 (2023), p. 103561.

[71] *BEHAVIORAL MODEL (bmv2)*. original-date: 2015-01-26T21:43:23Z. Apr. 2024. URL: https://github.com/p4lang/behavioral-model (visited on 12/19/2022).

[72] Elie Kfoury et al. *A Comprehensive Survey on SmartNICs: Architectures, Development Models, Applications, and Research Directions*. en. arXiv:2405.09499 [cs]. May 2024. URL: http://arxiv.org/abs/2405.09499 (visited on 06/19/2024).

[73] *Agilio® CX 2x10GbE SmartNIC*. URL: https://netronome.com/wp-content/uploads/PB_Agilio_CX_2x10GbE-3-24.pdf (visited on 04/20/2024).

[74] *Bypassing Firewall Rules*. URL: https://nmap.org/book/firewall-subversion.html (visited on 12/07/2022).

[75] *Chapter 5. Port Scanning Techniques and Algorithms | Nmap Network Scanning*. URL: https://nmap.org/book/scan-methods.html (visited on 12/06/2022).

[76] *ditg*. URL: https://traffic.comics.unina.it/software/ITG/ (visited on 03/28/2024).

[77] *FastReact*. URL: https://github.com/andrkass/FastReact/tree/master (visited on 11/29/2023).

[78] Andy Fingerhut. *A traffic manager with a progammable packet scheduler*. URL: https://forum.p4.org/t/a-traffic-manager-with-a-progammable-packet-scheduler/497/2 (visited on 08/05/2024).

[79] Andy Fingerhut. *Floating point operations*. URL: https://github.com/jafingerhut/p4-guide/blob/master/docs/floating-point-operations.md (visited on 01/17/2024).

[80] *Firewall | IDS Evasion and Spoofing*. URL: https://nmap.org/book/man-bypass-firewalls-ids.html (visited on 12/07/2022).

[81] *IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. URL: https://www.unb.ca/cic/datasets/ids-2018.html (visited on 01/12/2023).

[82] *Intel® Tofino™*. URL: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html (visited on 08/06/2024).

[83] *Intel® Tofino™ 2*. URL: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino-2.html (visited on 08/06/2024).

[84] *Internet Assigned Numbers Authority*. URL: https://www.iana.org/ (visited on 04/20/2024).

[85] *iperf3*. URL: https://iperf.fr/ (visited on 03/28/2024).

[86] *Mininet Overview - Mininet*. URL: http://mininet.org/overview/ (visited on 01/01/2023).

[87] *MITRE ATT&CK®*. URL: https://attack.mitre.org/ (visited on 04/20/2024).

[88] *Netronome NFP-4000 Flow Processor*. URL: https://d3ncevyc0dfnh8.cloudfront.net/media/documents/PB_NFP-4000-7-20.pdf (visited on 04/20/2024).

[89] *Nmap: the Network Mapper - Free Security Scanner*. URL: https://nmap.org/ (visited on 01/01/2023).

[90] *P4*. URL: https://opennetworking.org/p4/ (visited on 02/09/2023).

[91] *P4 Workflow*. URL: https://p4.org/ (visited on 06/17/2024).

[92] *P4C*. URL: https://github.com/p4lang/p4c (visited on 06/17/2024).

[93] Paul Pols. *The Unified Kill Chain*. URL: https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf (visited on 06/17/2024).

[94] Komal Shah. *P4: Programming Networks Forwarding Plane*. URL: https://www.volansys.com/blog/p4-programming-networks-forwarding-plane/ (visited on 06/17/2024).

[95] *Snort - Network Intrusion Detection & Prevention System*. URL: https://www.snort.org/ (visited on 01/19/2023).

[96] *The Modbus Organization*. URL: https://modbus.org/ (visited on 01/16/2023).

[97] *The Zeek Network Security Monitor*. URL: https://zeek.org/ (visited on 01/16/2023).

[98] *V1Model*. URL: https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4 (visited on 06/10/2024).